HW4

2017030055 윤상현

이번 문서에서는 mete의 static analyze tool인 clang을 사용한 static analyze를 통해 3개의

Task 코드를 분석하고, 그들의 True positive, False positive, False negative를 확인해보고, 이들의 의미와 이유에 대해 분석해보겠습니다.

Task 1

Tase 1은 True Positive에 대한 분석입니다.

Static analyzer는 clang을 사용하여 검출하였습니다.

True Positive는 static analyze가 올바른 에러를 report한 상황, 즉 static analyzer가 성공적을 역할을 완료했음을 의미합니다. 제가 구현한 task code는 아래와 같습니다.

```
#include<iostream>
using namespace std;
int main(){
        int x=10;
        int y=1;
        y=y-1;
        x=x/y;
        cout<<x;
}
```

두 정수형 변수 x, y에 각각 10, 1을 선언하고, y의 값을 1 내려 0으로 만듭니다.

이후 변수 x에 x/y를 수행하는데, 이는 실제로는 10/0의 연산이 됩니다.

하지만 0으로 숫자를 나누는 것은 불가능하고, 이는 보고 되어야할 에러입니다.

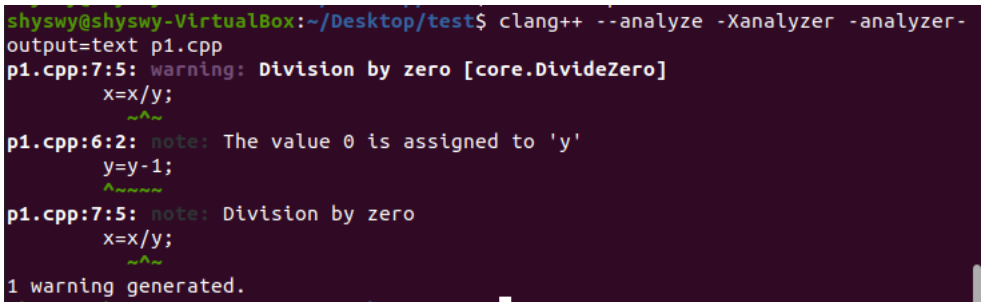
따라서 만약 해당 코드를 직접 실행한다면, 아래와 같은 에러가 검출되어야 합니다.

```
shyswy@shyswy-VirtualBox:~/Desktop/test$ ./p1
Floating point exception (core dumped)
```

Floating point exception은 divide by zero error를 의미합니다.

이를 통해 실제 실행 시, 0으로 특정 숫자를 나누게 되면, 에러가 되어 코드 전체를 손상시킬 수 있다는 것을 나타냅니다.

이번엔 clang을 통한 static analyze로 해당 코드를 분석해보았습니다.



위는 clang을 통해 static analyze된 화면입니다. 변수 x를 y로 나누는 부분에서, Division by zero Warning이 검출된 것에서 제대로 된 analyze가 이루어졌음을 확인할 수 있습니다.

이처럼 True Positive는 치명적이고 확실한 에러를 검출하는 것을 말합니다.

Static analyzer는 실제 코드가 완성되지 않거나, 전체적인 코드를 체크할 때 Dynamic analyzer에 비해 이점이 있습니다. 만약 방대한 양의 코드에서, static analyzer 없이 온전히 실행 결과에 의지해야 한다면, floating point exception이라는 exception의 종류 하나만 가지고 코드 전체를 디버깅하는 cost efficiency 하지 못한 작업을 반복해야 합니다. 하지만 static analyzer를 통한 성공적인 분석 결과인 True Positive는 이러한 상황에서 적절한 에러를 검출하도록 도와주고, 이는 전체적인 코드를 검사할 때 좋은 기능입니다.

Task2

Tase 2은 False Positive에 대한 분석입니다.

Static analyzer는 clang을 사용하여 검출하였습니다.

False Positive는 static analyze가 실제로는 에러가 아닌 영역을, 에러로 치부하고 report하는 것을 의미합니다. 이는 static analyzer가 주어진 역할을 성공적으로 완수하는 것에 실패했다는 것을 의미합니다. 아래는 task 코드입니다.

```cpp
#include <map>
#include<iostream>
using namespace std;
class DivZeroCheckerTestClass {
private:
    std::map<int, int> map;
    int num = 0;

public:
    DivZeroCheckerTestClass(int num) {
        this->num = num;
    }

    int func(int a) {
        if (a == num)
            return 0;

        map.clear();//this cause false positive!

        if (a != num)
            return -1;
        return 10 / (a - num);
    }
    int chk() { return num;}

};
int main(){
        DivZeroCheckerTestClass a=  DivZeroCheckerTestClass(5);
        a.func(4);
        cout<<a.chk()<<endl;
//      cout<<a.chk2()<<endl;


}
```

DivZeroCheckerTestClass라는 class는 생성자를 통해 정수형 변수를 parameter로 받습니다. 그리고

해당 변수를 private 정수형 변수인 num에 저장하게 됩니다.

해당 클래스 내부의 func() 함수는 정수형 parameter, a를 한 개를 받습니다. 이후, 자신이 속한 객체의

Private 정수형 변수인 num과 parameter로 들어온 a와 비교하여, 만약 둘이 같다면, 0을 return

합니다. 만약 둘이 다르다면, -1을 리턴하고, 그 외에는 10/(a-num)을 리턴 합니다.

위의 두 if문에서, 두 변수가 같다, 그리고 같지 않다 에 대한 return이 수행되었기에, 맨 아래의 return문이 실제로 실행될 일은 없고, 그렇기에 divide by zero문제가 생길 일은 실제로 없습니다.

main문에서는 예제로 생성자를 통해 객체의 private 변수 num을 5로 설정 후, func() 함수의 parameter에 숫자 4를 추가하여 a!=num의 케이스를 임의로 작성하였습니다. 이 경우, 두 값이 다르기에 return -1이 수행되고, 그 아래 역시 문제가 없습니다.

아래는 실제 실행 결과입니다.

```
shyswy@shyswy-VirtualBox:~/Desktop/test$ g++ -o p2 p2.cpp
shyswy@shyswy-VirtualBox:~/Desktop/test$ ./p2
5
```

아무런 에러가 검출되지 않았고, Chk() 함수를 통한 cout에서 올바르게 숫자를 출력하며 종료합니다.

아래는 static analyzer clang을 통한 분석 결과입니다.

```
shyswy@shyswy-VirtualBox:~/Desktop/test$ clang++ --analyze -Xanalyzer -outpu=tex
t p2.cpp
p2.cpp:22:19: warning: Division by zero [core.DivideZero]
        return 10 / (a - num);
                  ~~~^~~~~~~~~~
1 warning generated.
```

실제로는 실행되지 않은, 10/(a-num)에서 division by zero에러를 검출하였습니다.

이는 코드 중간에 들어간 map.clear()의 영향으로 추정됩니다.

이처럼, static analyzer의 한계로 인해 어쩔 수 없이 soundness, incompleteness 가 발생되는 경우가 존재합니다. 그중 이번 문서에서 다룬 false positive는 실제로 문제없이 동작하고, 에러로 치부되지 않지만, static analyzer가 오인하여 에러로 치부하는 실패적인 분석결과를 의미합니다.

Task3

Tase 3은 False negative에 대한 분석입니다.

Static analyzer는 clang을 사용하여 검출하였습니다.

False negative는 static analyze가 실제 에러를 report하지 못하는 것을 의미합니다.

아래는 task code입니다.

```
#include<iostream>
using namespace std;
int main(){
        int x=10;
        int y=10;
        while(y>=0){
                x=10/y;
                y=y-1;
                cout<<y<<" ";

        }
        cout<<x;
}
```

두개의 정수형 변수 x, y에 숫자 10을 할당합니다.

그리고 아래의 while문을 통해, y 값을 0까지 감소시킵니다.

X=10/y가 y=y-1 이전에 오기에, x=10/y의 라인에서 10을 나누게 될 y 값은 10~ 0이 될 것입니다.

따라서 실행결과는, 맨 마지막 while loop에서 divide by zero 에러를 검출하게 될 것입니다.

아래는 실제 실행 결과입니다.

```
shyswy@shyswy-VirtualBox:~/Desktop/test$ ./p3
Floating point exception (core dumped)
```

Floating point exception이 발생하며 실패했습니다.

이는 divide by zero 에러로 인한 exception입니다.

따라서 static analyzer가 올바르게 동작했다면, 똑 같은 에러를 report할 것입니다.

여기서 static analyzer가 에러를 검출하지 못한 이유를 추정해본다면, 아마 branch coverage가 100% 달성되지 못한 문제인 것 같습니다. while문의 가장 마지막에 divide by zero 에러가 검출되는 exception이 딱 1가지 경우의 수로써 존재하지만, 이것을 찾지 못해서 static analyzer는 에러가 없다고 판단한 것 같습니다.

이와 같이, 실제로는 에러가 있지만, 이를 찾지 못하는 것을 false negative라고 하며, 이는 static analyzer의 성공적이지 못한 결과입니다.

아래는 clang을 통한 static analyze 결과입니다.

```
shyswy@shyswy-VirtualBox:~/Desktop/test$ clang++ --analyze -Xanalyzer -outpu=te
xt p3.cpp
shyswy@shyswy-VirtualBox:~/Desktop/test$
```

실제와는 다르게, 아무런 에러를 검출하지 못한 채 종료되었습니다.

Reference

https://discourse.llvm.org/t/false-positives-of-clang-static-analyzer-divzerochecker-in-llvm14-0-6/64146