CS6888 Software Analysis HW2 report

Jianqiang Chen(jc3ze)

I. Program benchmark design considerations.
For each of the 10 programs, explain the program feature considered and justify why it was selected in terms of diversity and representativeness.

1. Simple Decision Make(result is good)
   Various decision could make different path for the symbolic execution. The example doesn't try the switch statement. So I plan to write a switch statement to test Klee's coverage ability, especially in multiple(more than 5) decision situation.

2. Simple Recursion (result is good)
   Recursion is special and wide use in programming. Every time the function enter its self will make increase branches number. I want to test if all branch of re-enter the recursion will be covered.

3. Simple Pointer (result is good)
   Pointer is a special structure in C/C++ compared with Java and Python. I want to test if the Klee will correctly find the address and cover that branch or create a branch.

4. Function (result is good)
   Will entering in a function will make Klee confuse? Just want to double check since function is a way to begin a new branch. Will it could pass symbolic variables to external function?

5. Loop (result is good)
   Loop is easily to create many new branches if it has decision make in the loop. I want to test if Klee could handle it.

6. Infeasible path (result is bad)
   As we discussed at class. The infeasible path is interesting. I want to test if the klee could report the infeasible path.

7. Malloc (result is bad)
   C program language could support allocate memory dynamically. If the klee cover the dynamic allocated address condition, it will very clever.

8. Struct (result is bad)
   Struct is unique in C. I want to test could klee only test some variables in the struct.

9. Typedef (result is bad)
   Typedef could be confusing to human sometimes. Will it also make Klee hard to find the real address?

10. Endless loop (result is bad)
    Will klee detect endless loop and stop it? Since it will create endless branches.

II. Create a table showing the number of tests and branch coverage achieved by Klee for each program in the benchmark (each program is a row in the table, the columns are program, # tests, and coverage). Make sure that each program clearly maps to one of the selected features discussed in the previous item.

Program	Tests	Branch Coverage(BCov(%))
1. Decision make	8	100%
2. Recursion	13	100%
3. Pointer	3	100%
4. Function	2	100%
5. loop	101	100%
6. Infeasible path	6	83.33%
7. malloc	3	75%
8. Struct	1	0.00%
9. Typedef	1	0.00%
10. Endless loop	N/A	Not finishing over 2 hours

III. Discuss your findings.
- Highlight and explain key observations revealed by your benchmark in terms of programming constructs that Klee handles either particularly well or badly.
- Were you surprised by Klee's handling or mishandling of any construct? Explain why or why not.
- For the constructs that are not handled well, conjecture why that is the case based on your understanding of symbolic execution.

What does Klee good at?

Based on the 5 good example, Klee could cover decision make, recursion, pointer & pointer function, external function and loop. I think it could symbolic analysis the grammar in the program. Based on the instruction, it could cover all the common path. In short, Klee could handle most of common used situation.

What does Klee not good at?

However, Klee is not perfect. It still will struggle in some situation:

- Endless loop

I notice that in the loop example, it takes klee abut 30s to cover 100 paths. And when the condition changed to 1000 paths, it will take more time. Initially I guess Klee should have some time limitation. However it runs over 2 hours and not give any error or result. So I think too many path will be a burden for Klee to deal with. Generate a new path takes klee more time than I expected.

- malloc example:

I think users should specific the valid range of memory allocation. It seems when the allocated memory is not valid, the Klee will report error and stop testing. Even after I add a decision statement to make the range valid, there is still a warning that Klee concertized the symbolic size. I find this in the Klee tutorial:
"*model*: *KLEE was unable to keep full precision and is only exploring parts of the program state. For example, symbolic sizes to malloc are not currently supported, in such cases KLEE will concretize the argument*."

I think Klee may not good at analysis dynamic allocated memory. If the memory size is not specific, the Klee cannot know the index of element in the array it want to use, then it could not continue the testing procedure. Even it succeed luckily in some testcase, it just report success tracks. And this will cause the Bcov not reasonable. In my code, there is 3 branches but the Bcov is 75%, which cannot happen. I searched the log, it only calculated the success branches and based on it to analysis.

- Infeasible path

I make two conflict conditions. Klee just covered all other paths. It makes sense. However, this case also should count as Klee cannot cover all paths. If the klee could give some warning while running, it should be better to use.

- Struct

KLEE: ERROR: object.c:15: memory error: invalid pointer: make_symbolic
In the case, I write a struct and want to pass symbolic variable to the struct. Just like this code:

```
Stu p1;
klee_make_symbolic(&p1.age, sizeof(p1.age), "p1.age");
printf( "Student age : %d\n", p1.age);
```

However, it could not work. I have read some wikis, the klee could accept the first address of variable. However, if just pass the following address, it could not recognize.

- Typedef

KLEE: ERROR: typedefName.c:15: memory error: invalid pointer: make_symbolic
KLEE: ERROR: 2.c:20: memory error: out of bound pointer
Typedef is an interesting case. It seems if the new typedef is not initialized, the typedef will not find the correct address even lots of testcases. And it also has problem when external call with symbolic argument. It not doing well when I use typedef in my way, and not finding document on how to deal with typedef.