CS 6110 — Assignment 4

Gavin Gray u1040250

1. (10 pts) Obtain KLEE's docker (as per Slides of Lec8). Go to the folder where sort.c is (in `klee_src/examples/sort`).
   - Comment out the `printf` statements – but read this: *largely follow the KLEE compilation instructions in the slides except I set some command-line options such as KLEEON; you can arrange those in which case this printf editing can be avoided.* Anyhow, attain the effects of printfs not being there.
   - Compile and run sort.c as in the slides. You will hit an assert violation.
   - Reinstate the prints (or arrange the KLEEON kind of tricks) and compile with gcc and run, feeding the file that failed the assert.
   - Can you diagnose the error using the test generated? Provide the details of your attempt to diagnose and document the findings.

Student Response: Running KLEE out of the box (i.e. no modifications to the source file `sort.c`) we can see that an assertion fails in a klee test as seen in 1.

```
KLEE: WARNING ONCE: Alignment of memory from call "malloc" is not modelled. Using alignment of 8.
KLEE: ERROR: sort.c:68: ASSERTION FAIL: temp1[i] == temp2[i]
KLEE: NOTE: now ignoring this error at this location

KLEE: done: total instructions = 16608
KLEE: done: completed paths = 24
KLEE: done: generated tests = 9
```

Figure 1: Failed Assertion

We can look into the `klee-last` directory to see the tests generated as pictured in 2.

```
klee@39355dde3229:~/klee_src/examples/sort$ ls klee-last
assembly.ll   run.istats            test000001.kquery  test000003.ktest  test000006.ktest  test000009.ktest
info          run.stats             test000001.ktest   test000004.ktest  test000007.ktest  warnings.txt
messages.txt  test000001.assert.err  test000002.ktest   test000005.ktest  test000008.ktest
```

Figure 2: 'ls klee-last/'

From the directory contents we can see that `test000001.ktest` was the one that produced the failed assertion. To debug, I'll compile the program with 'gcc' and enable print statements to try and narrow down the problem. The output of the run is pictured in 3.

```
input: [2, 16777216, 0, -1]
insertion_sort: [-1, 0, 2, 16777216]
bubble_sort   : [2, 0, -1, 16777216]
a.out: sort.c:71: test: Assertion `temp1[i] == temp2[i]' failed.
Aborted (core dumped)
klee@39355dde3229:~/klee_src/examples/sort$
```

Figure 3: test000001.ktest output

We can see that the output of the 'bubble_sort' routine is not properly sorted. Specifically, the member '-1' only "bubbles down" one position. This leads me to believe that a condition is missing whether to continue bubbling is something was swapped. Inspecting the code shows us just that!

```
void bubble_sort(int *array, unsigned nelem) {
  for (;;) {
    int done = 1;

    for (unsigned i = 0; i + 1 < nelem; ++i) {
      if (array[i+1] < array[i]) {
        int t = array[i + 1];
        array[i + 1] = array[i];
        array[i] = t;
        done = 0;
      }
    }
    // we're always breaking after 1 iteration!
    break;
  }
}
```

Figure 4: Bubble Sort Routine

In place of the comment before the `break;` statement, we can replace that with the following code

```
...
if(done)
  break;
...
```

Now the code never fails an assertion!

2. (20 pts) Write a routine to generate a symbolic array of characters of size 8, and calculate the max character in the array. Use any simple algorithm (e.g., walking the array linearly, keeping the "max so far."). Run this under KLEE, applying the kind of assertions one might like to use for a simple program like this. Document well.

   Student Response: The full source code is provided below. I chose to write the max char function in a complicated manner (with the use of goto and pointer manipulation) to see if I could slide an incorrect program past Klee. In short, I couldn't. Initially, I tried having the pointer `array` go one byte past the end of the array, however, with one test Klee detected this invalid memory access and stopped the program. With the below source, Klee does not provide any failing tests and generated 128 tests.

```
#include "klee/klee.h"
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

unsigned max_convoluted(char *array, unsigned n) {
  char * stop = array + n, c = *array++;
  unsigned i = 0;
BEGIN:
  if(array >= stop)
    goto END;
  if(*array > c)
    i = n-(stop-array), c=*array;
  array++;
  goto BEGIN;
END:
  return i;
}

void test(char *array, unsigned nelem) {
  char *temp1 = malloc(sizeof(*array) * nelem);
#ifdef PRINTON
  printf("input: [%d, %d, %d, %d, %d, %d, %d, %d]\n",
         array[0], array[1], array[2], array[3],
 array[4], array[5], array[6], array[7]);
```

```
#endif
  memcpy(temp1, array, sizeof(*array) * nelem);
  unsigned ans1, ans2;
  ans1 = max_convoluted(temp1, nelem);
#ifdef PRINTON
  printf("convoluted: %d\n", array[ans1]);
#endif
  for (unsigned i = 0; i != nelem; ++i)
    assert(array[ans1] >= array[i]);
  free(temp1);
}

int main() {
  char input[8] = { 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h' };
  klee_make_symbolic(&input, sizeof(input), "input");
  test(input, 8);
  return 0;
}
```

3. (20 pts) Enter the following FOL queries from Manna's book using the Alloy modeling methods discussed during Lec8. Show that the Alloy results confirm what Manna claims about the formula being valid or non-valid. Put enough comments after every formula and after every "check" and "assert" so that I can conclude that you are following the details well. If the formula has free variables, apply the universal closure and check for validity and existential closure and check for satisfiability (thus for such formulae, you will be producing two checks per formula).

    1. $p_1(x) \Rightarrow q_1(y)$ – two cases here (univ and exist closures)
    2. $(\exists x : \exists y : p_2(x, y)) \Rightarrow (\exists x : p_2(x, x))$. Do both the given formula and negated formula like I do (A1 and nA1 in my Alloy model for example). If a formula is valid, the negation is unsat. The "check" command works directly by seeing if it could falsify the assert (if so, giving a counter-example). Discuss the counterexample after including a figure thereof, in each case where there was a counter-example, documenting the reasons..
    3. $(\exists y : \forall x : p_2(x, y)) \Rightarrow (\forall x : \exists y : p_2(x, y))$. Details as above.

    Student Response: All three of the Alloy encodings should be found included in the zip file under `q3-1.als`, `q3-2.als`, `q3-3.als`.

    1. For both the universal and existential closure cases it is fairly trivial to see that this statement in invalid. We can check this hunch with just a few simple lines of alloy code. Because both the variables x, y and the predicates $p1, q1$ are disjoint, there are innumerable counterexamples for these claims. For example, in figure 5, we can see that the space $S1$ (which corresponds to predicate $p1$) has 4 members, however, for no member of the universe U is the predicate $q1$ true.

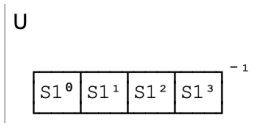

Figure 5: Counterexample for (a)

2. For this statement we again get a counterexample as seen in figure 6.

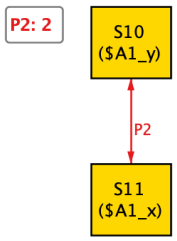

Figure 6: Counterexample for (b)

As we can see in the provided counterexample, $S0 \rightarrow S1$ is in the universe, as is $S1 \rightarrow S0$, however, as the predicate states, $S0 \rightarrow S0$ is *not* in the universe [1].

3. For the third part, alloy shows that the statement is valid, i.e. no counterexamples were found. We can however check the negation of the statement to see if the statement is unsat. This check provides a counter example (meaning that the statement is sat for some instances but not all), one such example is seen in figure 7.

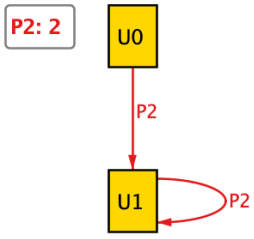

Figure 7: Counterexample to negation of (c)

4. (50 pts) Write a two-page project report as a Google Doc that is shared with me for placing comments. This report must contain the following information:
   - Name of project, project members.
   - Abstract telling what the project hopes to achieve.
   - Technologies/tools involved.
   - Other details, and a project flow diagram.
   - Create a project Github and mention it in the report.

   Student Response: The project report can be found at `https://docs.google.com/document/d/17ZZAjqLie0cPrQzehdTfCdwCTv15eYOVMhhGqkS2z4A/edit?usp=sharing` and the project GitHub page is found at the following: `https://github.com/gavinleroy/CS6110/tree/master/Proj` (also mentioned in the report).

[1] Note that $S0, S1 \in U$