Difficulties with T2 and Running Infer on Libuv and TheAlgorithms:Java

Dax Borde

1 Overview

While I had originally submitted the T2 Temporal Prover (https://github.com/mmjb/T2) as my term project idea, I ran into a number of issues getting it to work. As a result, I have switched to using the Infer (https://github.com/facebook/infer) static analyzer instead. I've used Infer to analyze of two different open source repositories: Libuv (https://github.com/libuv/libuv) and TheAlgorithms:Java (https://github.com/TheAlgorithms/Java). I had little success finding errors in both, for separate reasons. Repositories containing my results for each repo are located at https://github.com/daxborde/libuv-infer and https://github.com/daxborde/Java-infer/.

2 T2

I had previously proposed using the T2 Temporal Prover for this project, but ran into major difficulties. T2 is designed to prove termination properties of programs, namely whether they are terminating or non terminating. It was developed by Microsoft research, primarily due to their interest in verifying the termination of drivers for Windows machines. The primary issue I had with the project is that there are no existing binaries for the T2 analyzer. The only way to get a working version of the analyzer is to compile from the source code, which I was unable to achieve after many hours of trying.

2.1 Visual Studio

I first attempted to compile with Visual Studio 2019, the recommended method of compilation. I cloned the repository and attempted to build the Visual Studio solution file several times, getting one of 2 possible sets of errors each time. The first set of errors involved problems using the F# language runtime (the language the analyzer is written in), even after it was installed into Visual Studio. The second set of errors was made up of numerous syntax errors in the `IntervalIntDom.fs` file. These mostly complained about members of parameters in a function. (One line was `let mutable lr = l1.min l2`. `l1` is a parameter of the function being defined, and the compiler is complaining that `min` is not a valid member of it.) As I have little to no experience in F#, I was unable to debug these issues any further.

2.2 Linux Build Attempts

I also tried building on the Windows subsystem for Linux, which had its own issues. Compiling the source on Linux comes with its own challenges, as the Z3 solver (https://bitbucket.org/spacer/code/src/spacer/) must be compiled and included before T2 can be compiled. When running the make file for Z3, gcc ran into a number of errors, mostly due to variable names beginning with underscores. I tried the same build process on an Ubuntu VM through a tool called Vagrant (https://github.com/mmjb/T2/issues/22#issuecomment-312630057). This VM could not even clone the Z3 directory properly, consistently failing midway through the clone due to a git object being too large (the error was something along the lines of `fatal: pack has bad object at offset #####: inflate returned -5`). At around this point I decided that I had to try something else.

3 Infer

Since I had used it briefly in class before, I swapped over to the Infer analyzer. Infer is an open source static analysis tool that is primarily written and supported by Facebook. Infer's purpose is primarily to carry out general error checking before putting a system into a production environment. It is designed primarily to be useful and fast, rather than accurate, so "no strong claims" are made about the analyser's soundness and false positive rate. Infer can check a large variety of bugs, including dead stores, memory leaks, and thread deadlock. However, the errors I encountered most often were null dereferences and uninitialized value errors. Most of the errors I encountered were false positives, although the false positive rate differed greatly between the two projects.

3.1 Installing and Running Infer

Installing and Running the analyzer was simple and straightforward this time, as binary builds were available. I Installed the program through a downloaded tarball on the Windows subsystem for Linux. It is worth noting that using the subsystem was required for me, as the program does not appear to support the Windows platform. After adding the program to my path, the installation was successful. I am able to run Infer through the command line, and used a different set of commands for each open source repo.

4 Libuv

Libuv is an asynchronous I/O library written almost entirely in C. It was originally written to abstract away complicated network calls for the Node.js platform, but is used for a number of other languages as well. I used Infer on the project by running the following commands in the root of the repo:

```
mkdir build
cd build
infer compile -- cmake ..
infer run --keep-going -- make ./
```

Infer found 130 issues in this project. The full list of errors can be found at https://github.com/daxborde/libuv-infer/blob/v1.x/build/infer-out/bugs.txt. The breakdown of the errors is listed below:

```
Summary of the reports

     NULL_DEREFERENCE: 55
  UNINITIALIZED_VALUE: 43
        RESOURCE_LEAK: 14
          MEMORY_LEAK: 11
           DEAD_STORE: 7
```

I hand checked 27 of the issues myself, with admittedly limited knowledge of the repo code. I was able to eliminate 24 of the issues as false alarms, with 3 remaining as possible true alarms. It appears that most of the null dereferences and uninitialized values are false alarms. Most of these errors were the result of the odd systems-like programming of the library. The following is an example of a loop initialization that caused a null dereference issue:

```
uv_loop_t* loop = uv_default_loop();
```

The method being called seemed to always return an integer, but Infer was unable to determine this for certain, so any subsequent use of the `loop` variable caused a null dereference issue. Multithreading is also occasionally used in this repo, so whenever a child thread is expected to change a previously uninitialized variable, an uninitialized value issue is displayed. The 3 issues that remained plausible were all memory leak issues. There were a few methods that initialized memory with `malloc()` and never seemed to free it. However, all of the memory leaks were in the unit test section of the repo, so the code that actually ships doesn't have these issues.

5 TheAlgorithms:Java

TheAlgorithms:Java is a very different kind of open source project. It is essentially a large list of core computer science algorithms written in Java. The repo is mostly used for reference, and does not aim to ship a product of any kind. After testing Infer on a hardcore systems library written in C, it seems appropriate to provide contrast with a clean and compartmentalized Java repo. I ran Infer on each java file in the repository individually by running the following command in any folder with `.java` files:

```
infer run -- javac *.java
```

The full results of each run are combined in this file: https://github.com/daxborde/Java-infer/blob/master/infer-all-outputs.txt. The vast majority of the implementations in the repo had no issues at all, in stark contrast to the Libuv repo. However, three issues were found in `Fibonacci.java`. All 3 were null dereference issues, and they were all detected in the function below:

```java
public static int fibBotUp(int n) {
  Map<Integer, Integer> fib = new HashMap<>();

  for (int i = 0; i <= n; i++) {
    int f;
    if (i <= 1) {
      f = i;
    } else {
      f = fib.get(i - 1) + fib.get(i - 2);
    }
    fib.put(i, f);
  }

  return fib.get(n);
}
```

The errors indicated that `fib.get()` may return `null` in certain circumstances. As the values passed to `fib.get()` will always have been inserted before the call, the issue is actually a false positive.

6 Discussion and Conclusion

Infer produced wildly different results on both repos. On the Libuv repo, the analyzer was often tricked into giving a false positive. The most common causes of this were functions that have odd side effects or return pointers. Using such functions seems to be common in C programs at the systems level, so perhaps Infer can be modified to deal with these odd paradigms in the future. The TheAlgorithms:Java repo gave different results entirely, with almost no issues being reported at all. This may be attributed to the much less complicated task being solved, and the more standard use of the chosen language. Overall, despite the lack of bug finding productivity and a small smattering of limitations, Infer proved to be an easy tool to install, set up, and run on various types of repositories.