Thuy Nhi Tran-Thien
Hang Bui

CSE 403 Project Pitch: RewinDB

Motivation

Standard debuggers do not offer the option of stepping back. Too often, programmers set the breakpoint too far or step through the debugger too fast and miss the step they wanted to examine and have to restart. For example, imagine a CS student who is trying to debug their program. They want to look at what is in their array at line 15, but they accidentally advance to line 16 when the array is discarded. With a reverse debugger, they could simply examine the state of the array by looking at the program state from line 15, which our debugger would store for reference.

Currently, reverse debuggers exist but are either too heavy (stores entire stackframe for the duration of the program) or do not allow live reverse-debugging (the entire program must execute to be recorded, then user can begin debugging). For most of us who are only interested in a few previous steps, the first option is overkill and may slow down the debugger. The second one is also limited because one may want to step through the program like in a regular debugger or only look at a certain part, not execute the entire thing.

Approach

Our reverse debugger would simply store the program state for a couple steps and allow the user to view these states during the debugging process. We liked the idea of "recording" the program as opposed to actually undoing the changes in order to catch irreproducible bugs and address concerns of irreversible execution such as network calls or file changes. When the developer wants to "step back", we would pause the program's execution and display the previously stored information. This gives the illusion of reverse debugging and achieves the goal of letting the developer see what just happened. Then, when the developer steps forward back to the place that he or she left off, the debugger resumes executing the program. With other solutions like Chronon, we would have to wait for the program to finish executing in order to replay the events. This would be time consuming if we only want to debug the beginning of the code. Thus, we want to modify this idea and incorporate the recording into the live debugging session. Additionally, to address the problem of a memory and speed, we only record recent states instead of the whole program. For example, we might choose to record only the last 10 lines in a cache and replace the least recently used data with new ones as we step through the program.

Thuy Nhi Tran-Thien
Hang Bui

While our approach does not allow the user to go back and examine any previous point in the program, often times we only miss the desired line by a few steps, so we feel our method is sufficient. It also increases the speed of our debugger, as opposed to storing all program states.This approach also allows the user to do live debugging by stepping backwards or forwards like a standard debugger, instead of simply recording the execution. A user may also want to change variables or do other tasks at runtime, which our reverse debugger would allow.

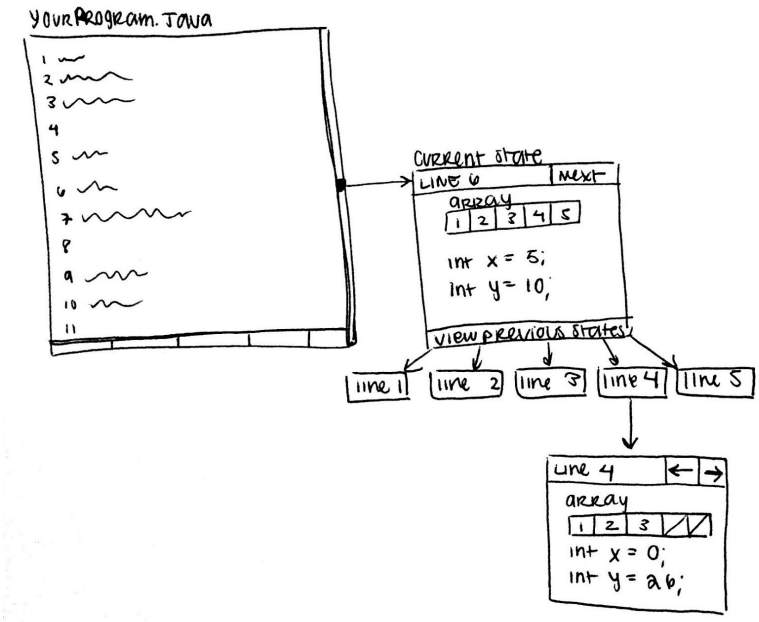

Figure 1: Rough sketch of our proposed reverse debugger

We think our reverse debugger could be useful for all programmers. Everyone needs to debug, and it is always more convenient to have the option to go back a few steps. However, for beginner programmers who are not as used to debuggers, it is more likely for them to overstep, so we feel our solution would be especially useful for them. Overall, this means less time and frustration spent on debugging. We can measure the performance of our reverse debugger by considering speed and memory use. Our goal is to create a lightweight debugger that will be more efficient and use less memory that the current solutions, while still being a sufficient reverse debugger.

Challenges & Risks

Although our idea seems simple enough, it may be difficult to actually implement. For example, we may run into roadblocks when trying to find efficient ways to store program states. There is also the question of what exactly we are going to store, and also designing the user interface. We also would like to implement features such as allowing the user to select how many previous steps they would like to save, which may also be a challenge to implement. However, we think that our solution is feasible. It will likely take the form of a plugin for a common IDE, such as Intellij or Eclipse. There have also been previous solutions to our problem that are heavier or more thorough than ours, so a more lightweight solution is definitely possible.

Time Spent: 3.5 hours