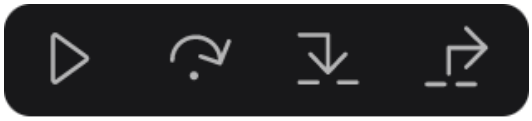
Neil's News

Backwards Debugging

28 October 2022

Every programmer will recognize these controls:



They are "Run", "Step over", "Step in", and "Step out" respectively. The obvious control that's missing is "Step back". When one encounters an error, it would often be quite useful to be able to wind back execution until the source of the error is discovered. Instead, debugging can be a frustrating exercise of trying to place a breakpoint ahead the error, but one that won't trigger a hundred times before the error manifests itself.

From a theoretical perspective, backwards execution is impossible. A statement as simple as `x = 0` cannot be reversed, since there's no way to know what the value of `x` was before the assignment (particularly if it was set using user input). But from a practical perspective, backwards execution is quite simple. Just save a dump of memory after every step. Then to go backwards, load the dump back into memory and the state at that moment will be restored. Any (non-quantum) programming language can do this.

I've created a demo using the JS-Interpreter of backwards stepping. After every forwards step it serializes the state and pushes that onto a stack. For every backwards step it pops the most recent serialization off of the stack, and restores the interpreter to that sate.

The main issue is that a full serialisation or memory dump can be quite large. In the case of JS-Interpreter it's about 300 KB. That's 1 GB every 3,000 steps. Fortunately this bloat is easy to get under control since each memory dump is virtually identical to its predecessor. So instead of a full dump, all one has to do is store the diff between each adjacent dump, thus forming a linked list. The result brings memory usage down to a few bytes per step.

I'm not sure why backwards debugging isn't a standard feature in IDEs. It's really quite trivial to implement.

< Previous | Next >