The Surprising Connection Between Memoization,Test Cases, Runs with Example Data, and the Method Finder

Ted Kaehler

VPRI Research Note RN-2017-002

Viewpoints Research Institute, 1025 Westwood Blvd 2nd flr, Los Angeles, CA 90024 t: (310) 208-0524

VPRI Research Note

The Surprising Connection Between Memoization, Test Cases, Runs with Example Data, and the Method Finder

by Ted Kaehler - June 2017

A Memo Function is a cache on the front of a method. When a set of argument values arrives that has already been seen before, the method replies immediately with the result it computed last time. A dictionary holds each set of arguments and a corresponding answer.

This memo about memoization describes some surprising benefits of memoizing a method.

1. **Speed**. A memoized method runs faster, in most cases.

2. **Automated Creation of Unit Tests**. The dictionary of arguments and answers is an excellent set of test cases. These can be automatically added to a testing suite, and they provides a much richer and broader set of tests than human-written Unit Tests.

3. **Automated Collection of Type Information**. The dictionary of arguments and answers provides type data for the method. The types of each argument and the answer are available. These are not complete -- some valid types may not have been collected in actual use so far. Researchers have used instrumentation to collect types in some systems. The advantage here is that this 'sketch' of type information comes for free, without any additional tools or measurements.

4. **A Method is Runable**. A programmer trying to understand a method can run it on an example from the memo dictionary. He can single-step it and watch it work.

With this capability, it is highly advisable to display the partial results beside every line of code. In addition, the user can execute any expression to see its value. The user can see the values of the variables. Having a working example makes this possible.

This is a form of documentation that takes no effort to create.
Why should a debugger provide a rich set of actual data values, and code in a library not provide them? The goal is to let the programmer see as much information when choosing a method, as he sees in a debugger at runtime.

5. **Builds a Method Finder**. A Method Finder, such as the one in Squeak, lets the user search for a method without knowing its name. The user types arguments and an answer. The Method Finder searches the library for a method that gets the right answer.

Currently, an expert has to decide which methods can be included in the search. A method must have no side effects, know if it modifies its arguments, and do no damage.

Exactly the same criteria apply to methods that can be memoized. They must be side-effect free. If they are not, just returning the correct answer would fail to preform the side-effect action of the method. The programmer knows this when he examines a method to see if he should memoize it. Instead of trying to convince a person to review the safety of a method that he may never use, the person now does the same review for a method he cares about and will definitely use. He gets the reward of a faster program after the method is memoized.

When a programmer memoizes a method, that method is automatically added to the list of approved methods for the Method Finder.

We have seen five benefits of memoizing a method. In many systems, the user must write his own code to manage the memo dictionary. Here are some tools that would encourage programmers the memoize their methods.

A. **One-Click Memoization**. Just press the button, and the memo mechanism is created at the front of a method.

B. **One-Click Verification**. A per-method switch that runs **both** the body of the method, and looks up the memoized answer. It throws an error if the answers are different. The method continues to run with verification on until the programmer is sure that memoization has not changed anything.

What can make a method ineligible to be memoized? Side effects include modifying a global variable, or changing something inside of a class. It should not change any values inside of its arguments. It should not close connections or remove things from global tables. It must not call a subroutine several levels down, in a special case, that does any of the above. These are subtile issues and require deep thought.

C. **Automatic Speedup Statistics**. For each method with One-Click Verification turned on, take speed data. Measure the time for both running the method and for looking up the value. Save those stats and display them whenever the user looks at the method in the library.

D. **Verification Toggle**. There is a single switch that can disable/enable verification for all methods that are currently marked for verification. When the user decides that the system seems to be working, run for a while with verification off. If something goes wrong, rerun with it turned on to see if any memoized method is at fault.

E. **Master Switch**. Temporarily disable memoizing for all methods (even if they were not being verified). When tracking down difficult bugs, do this to eliminate memoization as a possible source of error.

F. **Add to the Method Finder**. As mentioned above, every time the user decides to memoize a method, it is automatically added to Method Finder. That method can later be discovered by example by a different user. This is especially useful when the user does not know the name of the method or what library it belongs to.

There needs to be a way to share the newly approved methods. Other instances of the Method Finder will check a shared repository and update their list of approved methods.

It is highly advisable to add these tools to every IDE. Clojure, Haskell and other languages encourage code to be side-effect free. Such methods can be memoized easily. Other languages will also benefit from these tools.

Making a method in a library **runnable** deserves special attention (#4 above). When a programmer looks at a method and does not understand what it does, there is nothing as valuable as seeing it run line-by-line on a real example. Having the dictionary of memo data with sets of actual arguments, allows this to happen.

The Squeak Method Finder is 18 years old. During that time, this author has searched for automated ways to determine which methods could be used for trial evaluation by the Method Finder. Now there is a solution! Provided with the right tools, programmers will happily do the work of deciding which methods can be memoized. These can be added to the repertoire of the Method Finder, allowing them to be found by future users. In addition, the Method Finder will execute faster because the methods it tries are memoized.

This connection between memoization and the Method Finder is surprising and very useful.