2-1-2022

Unreachable Statements Are Inevitable In Software Testing: Theoretical Explanation

Francisco Zapata
The University of Texas at El Paso, fcozpt@outlook.com

Eric Smith
The University of Texas at El Paso, esmith2@utep.edu

Vladik Kreinovich
The University of Texas at El Paso, vladik@utep.edu

Comments:
Technical Report: UTEP-CS-22-20

Unreachable Statements Are Inevitable In Software Testing: Theoretical Explanation

Francisco Zapata, Eric Smith, and Vladik Kreinovich

Abstract Often, there is a need to migrate software to a new environment. The existing migration tools are not perfect. So, after applying such a tool, we need to test the resulting software. If a test reveals an error, this error needs to be corrected. Usually, the test also provides some warnings. One of the most typical warnings is that a certain statement is unreachable. The appearance of such warnings is often viewed as an indication that the original software developer was not very experienced. In this paper, we show that this view oversimplifies the situation: unreachable statements are, in general, inevitable. Moreover, a wide use of above-mentioned frequent view can be counterproductive: developers who want to appear more experienced will skip potentially unreachable statements and thus, make the software less reliable.

1 Unreachable Statements Happen

Many software systems periodically migrate to new software environments:

- new operating systems,
- new compilers,
- sometimes even new programming language.

Usually, most of this migration is performed automatically, by using special migration-enhancing tools: without such tools, migration of a million-lines code would not be possible.

However, the result is rarely perfect. For example, some tricks that use specific feature of the original operating system may not work in the new software environment. So, before using the migrated code, it is important to test the result of

Francisco Zapata, Eric Smith, and Vladik Kreinovich
University of Texas at El Paso, El Paso, Texas 79968, USA
e-mail: fcozpt@outlook.com, esmith2@utep.edu, vladik@utep.edu

automatic migration. Compilers and other testers test the migrated code and produce errors and warnings; see, e.g., [1, 3, 4].

- Errors clearly indicate that something is wrong with the program. Once a compiler or a tester find an error, this error needs to be corrected.
- In contrast, a warning does not necessarily mean that something is wrong with the program: it may simply mean that a software developed should double-check this part of the code.

One of the most frequent warnings is a warning that a certain statement is unreachable. Such warnings are the main object of this study.

2 Unreachable Statements: Why?

The ubiquity of unreachable statements prompts the need to explain where they come from. An unreachable statement means that:

- the original software developer decided to add a certain statement to take care of a situation when certain unusual conditions are satisfied, but
- the developer did not realize that these conditions cannot be satisfied – while the newly applied compiler or tester detected this impossibility.

3 Can We Avoid Unreachable Statements?

As we have mentioned, the main reason for the appearance of unreachable statements is that the original software developer(s) did not realize that the corresponding conditions are never satisfied. This implies that a more skilled developer would have been able to detect this fact and avoid these statements. So maybe if we have sufficiently skilled developers, we can avoid unreachable statements altogether?

Unfortunately, a simple analysis of this problems shows that, in general, it is not possible to detect all unreachable statements – and thus, that unreachable statements are inevitable.

Indeed, the condition that need to be satisfies to get to this statement is often described as a boolean expression, i.e., as an expression formed by elementary true-false conditions $c_1, \ldots, c_n$ by using boolean operations "and" (&), "or" ($\vee$), and "not" ($\neg a$). For example, we can have a condition $(c_1 \vee c_2)\,\&\,(\neg c_1 \vee \neg c_2)$.

If the corresponding boolean expression is always false, then the condition is never satisfied and thus, the corresponding statement is unreachable.

There are boolean expressions which are never satisfied. For example, the following boolean expression is always false:

$$(c_1 \vee c_2)\,\&\,(c_1 \vee \neg c_2)\,\&\,(\neg c_1 \vee c_2)\,\&\,(\neg c_1 \vee \neg c_2).$$

One can easily check that this expression is always false by considering all four possible combinations of truth values of the boolean variables c_1 and c_2:

- both c_1 and c_2 are true,
- c_1 is true and c_2 is false,
- c_1 is false and c_2 is true, and
- both c_1 and c_2 are false.

The problem of checking whether a given boolean expression is always false is known to be NP-hard; see, e.g., [2]. This means, crudely speaking, that unless P = NP (which most computer scientists believe to be false), no feasible algorithm is possible that would always provide this checking. Not only this problem is NP-hard, it is actually historically the first problem for which NP-hardness was proven. (To be more precise, the historically first NP-hard problem was to check whether a given boolean expression E is always true, but this is, in effect, the same problem, since an expression E is always false if and only if its negation $\neg E$ is always true.)

This NP-hardness result means that no matter what testing algorithm the software developer uses, for a sufficiently large and sufficiently complex software package testing will not reveal all unreachable statements. In other words, unreachable statements are inevitable.

To be more precise:

- a novice software developer may leave more such statements in the code;
- an experienced software developer will leave fewer unreachable statements;

however, in general, unreachable statements are inevitable.

4 So Can We Use the Number of Detected Unreachable Statements to Gauge the Experience of the Original Software Developer?

At first glance, the conclusion from the previous section seems to be that we can use the number of detected unreachable statements to gauge the experience of the original software developer:

- if the migrated software has a relatively large number of detected unreachable statements, this seems to indicate that the original software developer was not very skilled;
- on the other hand, if the migrated software has a relatively small number of detected unreachable statements, this seems to indicate that the original software developer was more skilled.

But can we really make such definite conclusions?

As we have mentioned earlier, the main reason why unreachable statements appear is that a software developer is not sure whether the corresponding condition is possible – and, as we have mentioned in the previous section, no feasible algorithm

can always check whether the given condition is possible. So, if we start seriously considering the number of detected unreachable statements as a measure of the quality (experience) of the original software developer, developers who want to boost their reputation would have a simple way of increasing their perceived quality: if we do not know whether a condition is possible or not, just do not add any statement for this condition. This, by the way, will make the program more efficient, since there will be no need to spend computer time checking all these suspicious statements.

In this case, the number of detected unreachable statements will be 0, so, from the viewpoint of this criterion, the developer will look very experienced. But what if one of these conditions is actually satisfied sometimes? In this case, for this unexpected condition – for which we did not prepare a proper answer – the program will produce God knows what, i.e., we will have an error. Is this what we want?

We have made the program more efficient, faster to run, but it is now less reliable. Is a small increase in efficiency indeed so important that we can sacrifice reliability to achieve it? Not really:

- With most modern computer application, small increases in running speed – e.g., time savings obtained by not checking some conditions – do not add any useful practical features: e.g., whether processing a patient's X-ray takes 60 seconds or 55 seconds does not make any difference.
- On the other hand, reliability *is* a serious issue. For example, if the software misses a disease clearly visible on an X-ray, we may miss a chance to prevent this disease evolving into a more serious (possibly deadly) condition.

From this viewpoint, the presence of unreachable statements is, counter-intuitively, a good sign: it makes us more confident that the program is reliable.

Acknowledgments

This work was supported in part by the National Science Foundation grants 1623190 (A Model of Change for Preparing a New Generation for Professional Practice in Computer Science), and HRD-1834620 and HRD-2034030 (CAHSI Includes), and by the AT&T Fellowship in Information Technology.

It was also supported by the program of the development of the Scientific-Educational Mathematical Center of Volga Federal District No. 075-02-2020-1478, and by a grant from the Hungarian National Research, Development and Innovation Office (NRDI).

References

1. G. Blokdyk, *Software Modernization: A Complete Guide*, The Art of Service, 2020.
2. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, Cambridge, Massachusetts, 2022.

3. F. Zapata, O. Kosheleva, and V. Kreinovich, “How to Estimate Time Needed for Software Migration", *Applied Mathematical Sciences*, 2021, Vol. 15, No. 1, pp. 9–14.
4. F. Zapata, O. Lerma, L. Valera, and V. Kreinovich, “How to Speed Up Software Migration and Modernization: Successful Strategies Developed by Precisiating Expert Knowledge", *Proceedings of the Annual Conference of the North American Fuzzy Information Processing Society NAFIPS’2015 and 5th World Conference on Soft Computing*, Redmond, Washington, August 17–19, 2015.