Blame for Null (Artifact)

Abel Nieto
University of Waterloo, Canada
anietoro@uwaterloo.ca

Marianna Rapoport
University of Waterloo, Canada
mrapoport@uwaterloo.ca

Gregor Richards
University of Waterloo, Canada
gregor.richards@uwaterloo.ca

Ondřej Lhoták
University of Waterloo, Canada
olhotak@uwaterloo.ca

Abstract

This artifact is a companion to the paper "Blame for Null", where we formalize multiple calculi to reason about the interoperability between languages where nullability is explicit and those where nullability is implicit. Our main result is a theorem that states that nullability errors can always be blamed on terms with less-precise typing; that is, terms typed as implicitly nullable. We summarize our result with the slogan *explicitly nullable programs can't be blamed.* The artifact consists of a mechanized Coq proof of the results presented in the paper.

2012 ACM Subject Classification Software and its engineering → General programming languages; Theory of computation → Type theory; Software and its engineering → Interoperability; Theory of computation → Operational semantics

Keywords and phrases nullability, type systems, blame calculus, gradual typing

Digital Object Identifier 10.4230/DARTS.6.2.10

Funding This research was supported by the Natural Sciences and Engineering Research Council of Canada and by the Waterloo-Huawei Joint Innovation Lab.

Acknowledgements We would like to thank the anonymous reviewers for their valuable feedback.

Related Article Abel Nieto, Marianna Rapoport, Gregor Richards, and Ondřej Lhoták, "Blame for Null", in 34th European Conference on Object-Oriented Programming (ECOOP 2020), LIPIcs, Vol. 166, pp. 3:1–3:28, 2020. `https://doi.org/10.4230/LIPIcs.ECOOP.2020.3`

Related Conference 34th European Conference on Object-Oriented Programming (ECOOP 2020), November 15–17, 2020, Berlin, Germany (Virtual Conference)

1 Scope

The paper's contributions are reproduced below (the wording is adapted from the paper):

- A core calculus, *lambda null*, that formalizes the essence of type systems with implicit and explicit nullability, like those of Kotlin and Scala.
- A higher-level calculus, *stratified lambda null*, that models the interoperability between languages with implicit nullability and languages with explicit nullability.
- A metatheory for lambda null, consisting of the standard progress and preservation lemmas, as well as well as blame theorems that characterize how nullability errors can occur.

- A metatheory for stratified lambda null with two main components. First, a desugaring semantics for stratified lambda null. Second, our main result, which states that nullability errors can always be blamed on terms with less-precise typing; that is, terms typed as implicitly nullable. In the style of earlier work on blame, we summarize our result with the slogan *explicitly nullable programs can't be blamed.*

The artifact substantiates the contributions by providing a mechanized Coq proof of all results presented in the paper.

2 Content

The artifact is packaged as a Virtual Box VM (an `.ova` file) containing:
- A detailed guide listing the correspondence between definitions and lemmas in the paper and those in the mechanization.
- A mechanized Coq proof that includes all results from the paper.

3 Getting the artifact

The artifact endorsed by the Artifact Evaluation Committee is available free of charge on the Dagstuhl Research Online Publication Server (DROPS). In addition, the artifact is also available at: `https://github.com/abeln/null-calculus`.

4 Tested platforms

We have tested the VM with VirtualBox Version 6.1.6 r137129 (Qt5.6.3). The OS installed in the VM is Ubuntu 18.04.

5 License

The artifact is available under an MIT license.

6 MD5 sum of the artifact

0488c9d847dc6c4508e1e049f6768e9a

7 Size of the artifact

6.4 GiB