THE UNIVERSITY OF QUEENSLAND

AUSTRALIA

Project Proposal

The Title of Your Thesis

by

Achmad Afriza Wibawa

47287888

a.wibawa@uqconnect.edu.au

Electrical Engineering and Computer Science

School of ~~Information Technology and Electrical Engineering,~~

University of Queensland

Supervisor

Brae J. Webb, Mark Utting, Ian J. Hayes

School of ~~IT & Electrical Engineering~~ EECS

{B.Webb,M.Utting,Ian.Hayes}@uq.edu.au

21 September 2023

Contents

1 Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

2 Introduction

frequently underspecified?

Compilers are inherently inaccurate. This is due to the fact that: common pitfalls of the language it's written in ~~is~~ are *just* accepted by the community; edge cases of the language are not considered well ~~enough~~; or simply making a mistake inside the implementation [1, Sec. 1.2]. Human mistakes are natural in human-made softwares. As such, it is critical to *try* to minimize the intrinsic risks of errors ~~happening~~ in compilers.

Minimizing the risks of error is non-trivial. A suite of testing mechanisms are needed in order to ensure the reliability of a software. There are several ways to do this. For softwares, regression testing in the form of Unit, Integration, and System level tests are the industry standard ways for mitigating risks [2]. Such testing suites are ideal for softwares with ~~an~~ human understandable behavior. However, the behaviors of compilers itself are not exactly human-readable. As such, manually defining the obscure behaviors of compilers are tedious and time consuming [3].

Another way to verify the behavior of compilers is to **formally** ~~define~~ specify them [3]. This project follows up on previous works done to introduce formal semantics for GraalVM's [4] Intermediate Representation ~~through a Domain Specific Language (DSL) [5]~~, [6] ~~implemented~~ in Isabelle [7]. There are similar works done, i.e. CompCert [3] and Alive [8], [9]; all of which integrates the theoretical aspects of formal verification into the practicality of using it in a production setting.

intro DSL

This project ~~would~~ attempts to bridge ~~the~~ a subset of the gap between the formal semantics of GraalVM and integrating it into GraalVM's test suite [6], focusing on creating an Automated Testing Framework for GraalVM ~~IR~~ Optimization's DSL. The framework would represent an automated unit test generation and, *if possible*, automated simple proof generation. This would make it easy for GraalVM's developers to use the tool *as you go*, without being a *"proof expert"* on Isabelle.

To implement an Automated Testing Framework for GraalVM ~~IR~~ optimization's DSL, there are several options for the project to explore (in order of ideal solutions):

1. Utilizing `Isabelle Server - Client` interactions [10, Ch. 4] to generate test suite and simple proofs [11]–[14];

2. Extend the system of `Isabelle/Scala` to utilize the full functionality of Isabelle [10, Ch. 5];

3. Creating an interpreter for the DSL, and applying a set of rules as ~~a~~ regression test suite.

below paragraphs are incorrect

However, verification that DSL matches the implementation of GraalVM would be out of scope of this project. It would represent the 3rd step of the compiler verification research thread [1, pp. 5], and perhaps a future direction in Veriopt.

Describe the sections

The semantics of the Graal IR are defined in Isabelle separately to the DSL.
The DSL is used for the abstract specification of optimization passes.

3 Background

3.1 Software Testing

3.1.1 Regression Testing

paraphrase more

In software engineering, the most commonly used method of software testing would be **regression testing**. Regression testing revolves around ~~determining~~ identifying parts of the original program `P` that is changed to `P'`. Program `P` would have their original test suite `T`, and developers would need to define additional `T'` for `P'`. Subset of `T'` would be executed to verify the behavior of the program. Thus, that would ~~prove~~ the correctness of the program. [2]

unclear

not a proff for our context

In practice, regression testing would be limited to the capabilities of humans to define the behavior of the program. As such, there is an inherent risk of bugs happening due to human mistakes; as it's possible that the defined behavior is not correct. Defining the complete set of behavior of the intended program through testing requires developers to spend a considerable amount of time to write tests manually [15]. As such, **random testing** is introduced to substitute humans with deterministic computer behaviors.

deterministic computer behaviors are plainly wrong

3.1.2 Differential Testing

change differentialTesting to randomTesting: rework this whole section

Differential testing is a suite of random tests generated by the computer to determine the correctness of the program based on a predetermined set of rules [15]. For example, [15, pp. 102-104] would generate test cases and execute them on programs which have a predictable result. For instance, if a program would deviate from the expected results, then the program would have undefined behaviors in them.

The difficulty of random testing lies in determining the set of rules to generate test cases. In [15], the test cases are generated by substituting the subset of the input to a random input. While this allows test cases to be generated quickly, programs would only need several ~~of~~ *"interesting values"* or edge cases to consider in their behavior. As such, a stronger test suite such as an **inference-based test generation** would be preferrable in this project.

3.1.3 Inference-based Exhaustive Testing

Exhaustive tests such as [11] takes into account the possible variable bounds of a program and converts it to a set of inference rules. For a program to be correct, all of the premises (P, Q) in the inference rules $(P \rightarrow Q)$ must be correct. As such, finding a counterexample would be as simple as determining if the bounds of a variable would result in a satisfiable $\neg(P \rightarrow Q)$. This could be extended even further by checking the inference rules on a SAT solver to find premises where the inference rules would be incorrect [12, Ch. 5]. Exhaustive searching allows developers to only focus on defining the behavior of the program, rather than defining tests to define the behavior.

3.2 Formal Verification of Compiler

If software code ~~are~~ is the recipe for system behaviors, then compilers would be the chef that puts it all together. Most people would assume that the behavior of compiled programs would match exactly ~~as~~ the ~~result~~ input program. However, this is usually not the case [3]. Chefs would have their own way of creating magical concoctions from a recipe, and so does a compiler. Not only does a compiler try to replicate system behaviors, it would try to make them faster in their own ways; i.e. adding optimizations or reducing unneeded behaviors. However, the original behavior of the program must be preserved in order to consider a compiler to be correct [3], [6], [8], [9].

Validating compiler correctness is not easy. While the correctness of a compiler can be validated by defining behaviors through regression testing, it would be time consuming to do and not exactly productive [3], [16]. There are multitude of ways that a compiler can go wrong [1, Sec. 1.2]; all of which have their own specific way of verifying correctness. For example, Compcert [3] tries to tackle all of the implementation and semantics errors inside a compiler (See 3.2.1) – creating a completely verified compiler for the C language ~~platform~~. Formally verifying compilers in the scale of Compcert would require vast amounts of time and resources, which projects ~~sometimes~~ often doesn't have.

As such, there are smaller scale projects such as Alive [8] & Alive2 [9] that ~~would~~ focus on behavior translation errors ~~occured~~ in LLVM's peephole optimizer (See 3.2.2). Veriopt tries to be even more specific, focusing only on the side-effect-free data-flow behavior optimizations that occurs in GraalVM [4]–[6] (See 3.2.3).

for now only side-effects are modelled & some control-flow optimizations are proved but the bulk of the opts are data-flow

3.2.1 Compcert

Compcert verifies that a compiler is correct through several steps:

1. With deterministic programs, a compiler would compile a source program to the produced program – in which both of the programs ~~would~~ must have the same behavior.

   This step is done by augmenting the compiler code with a *certificate* – code that carries proof that the behavior is exactly as intended [3, Sec. 2.2].

2. Compiler optimization phases must be accompanied by the formal definition of their intermediate representation (IR) semantics [3], [5].

3. Lastly, to formally verify each of the optimization phases, a compiler must either:

   (a) Prove that the code implementing the optimization is correct [3, Sec. 2.4].
   Veriopt uses this approach in verifying data-flow optimizations (See 3.2.3).

   (b) Prove that the unverified code produce the correct behavior in their translation [3, Sec. 2.4].
   Alive uses this approach in verifying LLVM (See 3.2.2).

how is Compcert implemented?

3.2.2 Alive

Alive tackles a subset of compiler verification by verifying that code optimization ~~behaviors~~ inside LLVM are correct [8]. For example, a compiler would optimize $(LHS = x*2) \sqsupseteq (RHS = x << 1)$ (RHS is a refinement of LHS). While this may seem trivial, there would be a lot of edge cases where the behavior translation might be incorrect; e.g. buffer overflows.

To verify that code optimizations are correct, Alive utilizes inference-based exhaustive testing (See 3.1.3) that allows the tool to encode machine code behaviors to inference rules. These inference rules are then passed to SMT solvers to check for their satisfiability (See 1). abstract?
If the code is proven to be correct, then it would mean that the underlying optimization code is correct.

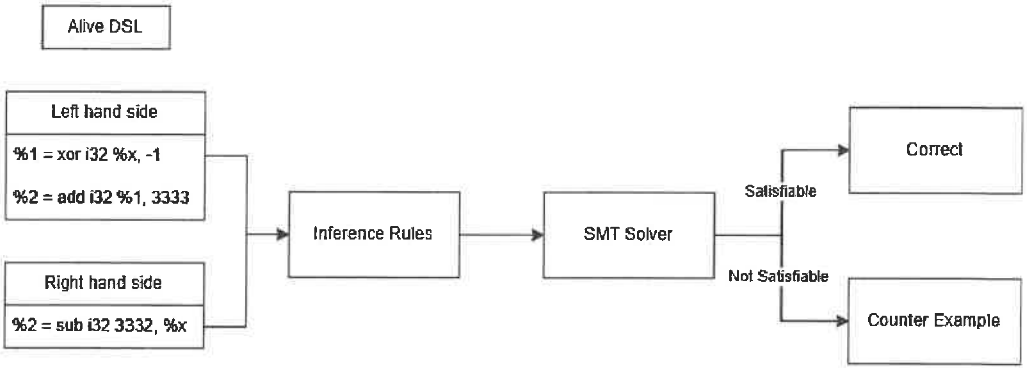

Figure 1: How Alive verif~~y~~ies $(LHS = ((x \oplus -1) + C) \sqsupseteq (RHS = X + (C - 1))$ [8, pp. 1]

Alive encodes LLVM's [17] underlying IR semantics through their own DSL [8, Fig. 1]. The DSL specification is made to be similar to LLVM's IR semantics to allow developers to easily integrate Alive with the development of DSL. This would represent the *certificate* that the underlying optimization is formally verified to be correct.

Alive takes this further by creating Alive2: a system to translate LLVM IR into Alive's IR [9]. This allows the developers to entirely focus on developing LLVM, while completely ignoring the specifications of Alive. This has been found to be effective, as differential testing of LLVM's unit tests and Alive2 discovers multiple errors inside the unit test behaviors itself [9, Sec. 8.2]. Alive & Alive2 would cover the whole 1st and 2nd step of compiler optimizations research thread [1, pp. 5].

3.2.3 Veriopt

In comparison to Compcert and Alive, the theoretical aspects of compiler verification are really similar. GraalVM's IR ~~are~~ is made up of two components: control-flow nodes and data-flow nodes. ~~All of~~ Which are combined as a 'sea-of-nodes' data structure [5]. However, Veriopt's DSL only concerns the subset of GraalVM's IR, which is the side-effect-free data-flow nodes [6]. Side-effect-free data-flow nodes are comparatively easier to prove and optimize, as it would be considered defined – as opposed to LLVM's undefined and poisoned variables [8], [9].

optimization *InverseLeftSub:* $(x - y) + y \longmapsto x$

Termination Proof Obligation $trm(x) < trm(BinaryExpr\,BinAdd(BinaryExpr\,BinSub\,x\,y)\,y)$

Refinement Proof Obligation $BinaryExpr\ BinAdd\ (BinaryExpr\ BinSub\ x\ y)\ y \sqsupseteq x$

Figure 2: Sample of Veriopt's DSL [6, Fig. 3]

fix this formatting

Figure 2 defines the structure that a side-effect-free expression would be optimized. unclear Note that there are 2 proof obligations that must be met in order to consider that the side-effect-free optimization is correct: proof that the optimization phase would terminate; proof that each pass of the optimization phase would result in a subset of the expression [6]. Note that these proofs would need to be provided by the users.

Currently, there's no existing tools that the developers of GraalVM could use to provide a *certificate* towards the compiler code [6, Sec. 7]. A semi-automated approach exists in a form of source code annotations [6, Sec. 5.1]. However, integrating new behaviors which would require new *certificates* would be challenging, as the approach would only describe the behavior of the code – instead of formally proving that the behavior is indeed correct.

Providing proof obligations for an optimization phase would be challenging for developers who are not *experts in program verification.* Veriopt's DSL are implemented in Isabelle [7], which comes with tools that assist in proving higher order logic (See 4.1). However, using such tools would require the developers of GraalVM to be familiar with Isabelle – something that ideally Veriopt would like to avoid. Similar tools such as Alive [8], [9] would be preferrable. Hence, that's where this project would like to contribute.

4 Methodology

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

4.1 Isabelle Overview

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

4.2 Utilizing Isabelle Server

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

4.3 Extending Isabelle/Scala

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

4.4 Interpreter for DSL

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

5 Project Plan

5.1 Milestones

5.1.1 Milestone 1: X

Milestone 1

Fusce tristique risus id wisi. Integer molestie massa id sem. Vestibulum vel dolor. Pellentesque vel urna vel risus ultricies elementum. Quisque sapien urna, blandit nec, iaculis ac, viverra in, odio. In hac habitasse platea dictumst. Morbi neque lacus, convallis vitae, commodo ac, fermentum eu, velit. Sed in orci. In fringilla turpis non arcu. Donec in ante. Phasellus tempor feugiat velit. Aenean varius massa non turpis. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae;

5.1.2 Milestone 2: Y

Milestone 2

Aliquam tortor. Morbi ipsum massa, imperdiet non, consectetuer vel, feugiat vel, lorem. Quisque eget lorem nec elit malesuada vestibulum. Quisque sollicitudin ipsum vel sem. Nulla enim. Proin nonummy felis vitae felis. Nullam pellentesque. Duis rutrum feugiat felis. Mauris vel pede sed libero tincidunt mollis. Phasellus sed urna rhoncus diam euismod bibendum. Phasellus sed nisl. Integer condimentum justo id orci iaculis varius. Quisque et lacus. Phasellus elementum, justo at dignissim auctor, wisi odio lobortis arcu, sed sollicitudin felis felis eu neque. Praesent at lacus.

5.1.3 Milestone 3: Z

Milestone 3

Vivamus sit amet pede. Duis interdum, nunc eget rutrum dignissim, nisl diam luctus leo, et tincidunt velit nisl id tellus. In lorem tellus, aliquet vitae, porta in, aliquet sed, lectus. Phasellus sodales. Ut varius scelerisque erat. In vel nibh eu eros imperdiet rutrum. Donec ac odio nec neque vulputate suscipit. Nam nec magna. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Nullam porta, odio et sagittis iaculis, wisi neque fringilla sapien, vel commodo lorem lorem id elit. Ut sem lectus, scelerisque eget, placerat et, tincidunt scelerisque, ligula. Pellentesque non orci.

5.2 Risk Assessment

Risk Assessment

Etiam vel ipsum. Morbi facilisis vestibulum nisl. Praesent cursus laoreet felis. Integer adipiscing pretium orci. Nulla facilisi. Quisque posuere bibendum purus. Nulla quam mauris, cursus eget, convallis ac, molestie non, enim. Aliquam congue. Quisque sagittis nonummy sapien. Proin molestie sem vitae urna. Maecenas lorem. Vivamus viverra consequat enim.

Nunc sed pede. Praesent vitae lectus. Praesent neque justo, vehicula eget, interdum id, facilisis et, nibh. Phasellus at purus et libero lacinia dictum. Fusce aliquet. Nulla eu ante placerat leo semper dictum. Mauris metus. Curabitur lobortis. Curabitur sollicitudin hendrerit nunc. Donec ultrices lacus id ipsum.

5.3 Ethics Assessment

Ethics Assessment

Donec a nibh ut elit vestibulum tristique. Integer at pede. Cras volutpat varius magna. Phasellus eu wisi. Praesent risus justo, lobortis eget, scelerisque ac, aliquet in, dolor. Proin id leo. Nunc iaculis, mi vitae accumsan commodo, neque sem lacinia nulla, quis vestibulum justo sem in eros. Quisque sed massa. Morbi lectus ipsum, vulputate a, mollis ut, accumsan placerat, tellus. Nullam in wisi. Vivamus eu ligula a nunc accumsan congue. Suspendisse ac libero. Aliquam erat volutpat. Donec augue. Nunc venenatis fringilla nibh. Fusce accumsan pulvinar justo. Nullam semper, dui ut dignissim auctor, orci libero fringilla massa, blandit pulvinar pede tortor id magna. Nunc adipiscing justo sed velit tincidunt fermentum.

Integer placerat. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Sed in massa. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Phasellus tempus aliquam risus. Aliquam rutrum purus at metus. Donec posuere odio at erat. Nam non nibh. Phasellus ligula. Quisque venenatis lectus in augue. Sed vestibulum dapibus neque.

References

[1] N. P. Lopes and J. Regehr, "Future directions for optimizing compilers," arXiv preprint, Sep. 2018. arXiv: 1809.02161 [cs.PL].

[2] N. J. Wahl, "An overview of regression testing," en, *ACM SIGSOFT Software Engineering Notes*, vol. 24, no. 1, pp. 69–73, Jan. 1999, ISSN: 0163-5948. DOI: 10.1145/308769.308790. [Online]. Available: https://dl.acm.org/doi/10.1145/308769.308790 (visited on 08/19/2023).

[3] X. Leroy, "Formal verification of a realistic compiler," en, *Communications of the ACM*, vol. 52, no. 7, pp. 107–115, Jul. 2009, ISSN: 0001-0782, 1557-7317. DOI: 10.1145/1538788.1538814. [Online]. Available: https://dl.acm.org/doi/10.1145/1538788.1538814 (visited on 08/19/2023).

[4] Oracle, *GraalVM: Run programs faster anywhere*, 2020. [Online]. Available: https://github.com/oracle/graal.

[5] B. J. Webb, M. Utting, and I. J. Hayes, "A formal semantics of the GraalVM intermediate representation," in *Automated Technology for Verification and Analysis*, Z. Hou and V. Ganesh, Eds., ser. Lecture Notes in Computer Science, vol. 12971, Cham: Springer International Publishing, Oct. 2021, pp. 111–126, ISBN: 978-3-030-88885-5. DOI: 10.1007/978-3-030-88885-5_8.

[6] B. J. Webb, I. J. Hayes, and M. Utting, "Verifying term graph optimizations using Isabelle/HOL," in *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs*, ser. CPP 2023, Boston, MA, USA: Association for Computing Machinery, 2023, pp. 320–333, ISBN: 9798400700262. DOI: 10.1145/3573105.3575673. [Online]. Available: https://doi.org/10.1145/3573105.3575673.

[7] T. Nipkow, M. Wenzel, and L. C. Paulson, *Isabelle/HOL: A Proof Assistant for Higher-Order Logic* (Lecture Notes in Computer Science). Berlin, Heidelberg: Springer, 2002, vol. 2283, ISBN: 3-540-43376-7. DOI: 10.1007/3-540-45949-9.

[8] J. Lee, C.-K. Hur, and N. P. Lopes, "AliveInLean: A verified LLVM peephole optimization verifier," in *Computer Aided Verification*, I. Dillig and S. Tasiran, Eds., Cham: Springer International Publishing, 2019, pp. 445–455, ISBN: 978-3-030-25543-5. DOI: 10.1007/978-3-030-25543-5_25.

[9] N. P. Lopes, J. Lee, C.-K. Hur, Z. Liu, and J. Regehr, "Alive2: Bounded translation validation for LLVM," en, in *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, Virtual Canada: ACM, Jun. 2021, pp. 65–79, ISBN: 978-1-4503-8391-2. DOI: 10.1145/3453483.3454030. [Online]. Available: https://dl.acm.org/doi/10.1145/3453483.3454030 (visited on 08/19/2023).

[10] M. Wenzel, "The Isabelle System Manual," en, url

[11] L. Bulwahn, "The New Quickcheck for Isabelle," en, in *Certified Programs and Proofs*, C. Hawblitzel and D. Miller, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2012, pp. 92–108, ISBN: 978-3-642-35308-6. DOI: 10.1007/978-3-642-35308-6_10.

[12] J. C. Blanchette, L. Bulwahn, and T. Nipkow, “Automatic Proof and Disproof in Isabelle/HOL,” en, in *Frontiers of Combining Systems*, C. Tinelli and V. Sofronie-Stokkermans, Eds., vol. 6989, Series Title: Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 12–27, ISBN: 978-3-642-24363-9 978-3-642-24364-6. DOI: 10.1007/978-3-642-24364-6_2. [Online]. Available: http://link.springer.com/10.1007/978-3-642-24364-6_2 (visited on 08/19/2023).

[13] J. Blanchette, “A Users Guide to Nitpick for Isabelle/HOL,” en, url

[14] J. Blanchette, M. Desharnais, and L. C. Paulson, “A Users Guide to Sledgehammer for Isabelle/HOL,” en, url

[15] W. M. McKeeman, “Differential Testing for Software,” en, vol. 10, no. 1, 1998.

[16] X. Yang, Y. Chen, E. Eide, and J. Regehr, “Finding and understanding bugs in C compilers,” in *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '11, New York, NY, USA: Association for Computing Machinery, Jun. 2011, pp. 283–294, ISBN: 978-1-4503-0663-8. DOI: 10.1145/1993498.1993532. [Online]. Available: https://dl.acm.org/doi/10.1145/1993498.1993532 (visited on 08/19/2023).

[17] *The LLVM Compiler Infrastructure Project.* [Online]. Available: https://llvm.org/ (visited on 08/20/2023).