″ALEXANDRU-IOAN CUZA″ UNIVERSITY, IAȘI

FACULTY OF COMPUTER SCIENCE

MASTER'S THESIS

Solidity Optimization with Control Flow Graphs

proposed by

Sergiu Iacob

Session: july, 2022

Scientific Coordinator

Dr. Arusoaie Andrei

″ALEXANDRU-IOAN CUZA″ UNIVERSITY, IAȘI

FACULTY OF COMPUTER SCIENCE

Solidity Optimization with Control Flow Graphs

Sergiu Iacob

Session: july, 2022

Coordinator

Dr. Arusoaie Andrei

Contents

List of Figures

Motivation

Titlu paragraf Blockchain has seen an increased popularity over the last few years. With more and more decentralized, scalable applications in the cloud, the software engineering industry has also seen an increase in blockchain development requests. It is futile to deny the revolutionary aspects of the Blockchain network, as it is usually the case with many new technologies in the market.

One of the keys for great software engineering is working with great, efficient tools – that is, the better, faster, more secure a programming language is, the better the final product is. In this thesis, we'll explore on how we can make Solidity, a programming language for developing smart contracts on the Blockchain, more efficient in building its bytecode and deploying the smart contracts in the network.

By the means of this work, we expect to see small improvements in the static analysis and compilation of the code, which in the end, multiplied by the hundreds of thousands of smart contracts compiled daily, might result in huge cost savings in the cloud.

Introducere

Solidity is a highly volatile language at this point. It might be that details presented in this thesis will lose veridicity in the future. However, the high-level schemas, approaches and structure of the Solidity eco-system will roughly be the same.

Solidity

YUL Intermediate Representation

Introduction

YUL IR is relatively new in the Solidity compiler.

First mention of YUL IR: 3rd of December, 2018, version 0.5.1.

Compiling via the YUL IR considered production ready with 0.8.13 release, March 16, 2022. Solidity's team focus is now on the YUL IR and they encouraged the usage of '–optimize –via-ir' at the Solidity Summit, April 2022 – as mentioned by Hari Mulackal.

Source: https://blog.soliditylang.org/category/releases/

Why is YUL necessary? * Figuring out optimizations on EVM bytecode is a hassle. Bytecode is not readable (lizibil) * YUL IR has a readable syntax – "medium" level, between Solidity and EVM bytecode * Generating tests is much easier – expected output vs actual output

https://docs.soliditylang.org/en/latest/yul.html

Obiective 1. Programs written in Yul should be readable, even if the code is generated by a compiler from Solidity or another high-level language.

2. Control flow should be easy to understand to help in manual inspection, formal verification and optimization.

3. The translation from Yul to bytecode should be as straightforward as possible.

4. Yul should be suitable for whole-program optimization.

Proprietati 1. Yul is statically typed (to avoid confusion between vals / references for example)

Asta de adaugat in alta sectiune

Bytecode based optimizer (din slide-uri Hari) ¿ works on basic blocks **??** (de prezentat control flow graph-urile inainte de asta) ¿ cannot really perform more complex optimizations ¿ De exemplu, https://www.youtube.com/watch?v=BWO7ij9sLuAlist=PLX8x
$SoliditySummit > minutul7 : 00, detranspusincuvinte > variabilaeOUTSIDEthebasicblock, pentr$

$ulbasicblock{-}uluieacelJUMPDEST > Itcould, buttheengineeringconsensisthatitshouldnot{-}TOO$
$OptimizingonYULismuch, mucheasier <-- seeHari's Summittalk(2022), 1stofMay$

Caveats of Optimizing ¿ Inline is currently an heuristic. ¿ Inline is awesome, but it can cause problems. The EVM can access only the first 16 stack slots. If you inline too much, it causes stack too deep issues. Balance between –optimize-runs

Alte avantaje ¿ function inlining – huge gas advantage, much easier to do in YUL

Control Flow Graphs

Solidity Optimizer

A way to compute the optimizer's efficiency

Daddy Yankee – gasolina.

Control Flow Graphs Study Case

Termination flows * stop() == return(0, 0) * return(p, s) – end execution, return data mem[p…(p+s)) * revert(p, s) – end execution, revert state changes, return data mem[p…(p+s))

Chapter 1

Solidity static analysis methods

Solidity already benefits from in-built static analysis tools and a compiler, *solc*. Some of the optimizations already done are well documented, as per Solidity's official documentation [**?**].

In this chapter, we'll go over some of the external tools built for Solidity optimization, as the main objective for this thesis is to extend, enhance and/or build such tools, as an improvement over the existing official toolkit.

1.1 Solidity Instrumentation Framework (SIF)

Built by Chao Peng, Sefa Akca and Ajitha Rajan, from University of Edinburgh, SIF is a tool built in C that gravitates around the Visitor Design Pattern. The main objective is to build an internal C representation of Solidity code, through intermediary data structures, upon which SIF can orchestrate various optimizations / refactoring functions. As the authors describe it, it's a tool "to easily and effectively understand, manipulate and analyse Solidity code" [**?**].

The drawbacks that this toolkit currently have is that it's outdated and that it does not allow for external users to "plug-in" their own intermediate representations of Solidity code. It acts more as a middle-man, allowing users to implement, through the *visit* method, specific optimization / refactoring items that run against isolated pieces of code. The tool does build a Control Flow Graph for the code, the one that this thesis will focus on, but it does not specifically use it for any kind of in-house optimization – it is just available to the user for usage.

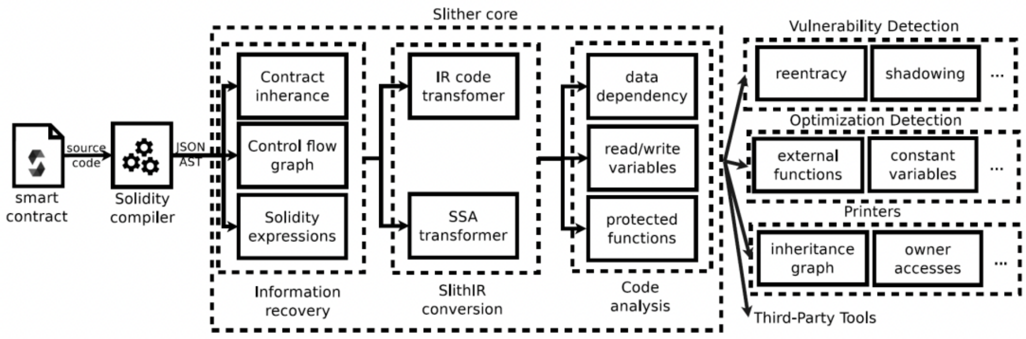

Figure 1.1: Slither Architecture

1.2 Slither

Slither is a more abstract, high level static analysis framework, which gives a broader overview over the internals of an Ethereum smart contract. Its main focus is centered around 4 areas: security (vulnerability detection), automated code optimization, code analysis and assisted code review, as described in Section 4.1 by the author Margherita Renieri [**?**].

The tool takes advantage of the Abstract Syntax Tree (AST) built by the Solidity Compiler, then passes that through a series of optimization steps: Information Recovery, SlithIR conversion and Code analysis.

What's common in both of the tools we've seen is the usage of Abstract Syntax Trees and Control Flow Graphs, as well as building "proprietary" intermediate representation of the input code.

Chapter 2

Contributions

Solidity is a relatively new technology, when compared to already existing programming languages widely used in the market. As stated by the Solidity team itself, ”the optimizer is under heavy development” [**?**], which gives plenty of room for contributors to bring their own performance enhancements.

I plan on being one of those contributors, starting from the Solidity code and the AST built by the Solidity compiler itself. I will then evaluate the given inputs and focus on building a specific, low resolution Control Flow Graph, by which memory allocation and cpu cycles can be thoroughly inspected. This would allow for a technical deep dive into the insides of an Ethereum smart contract, which in return can provide useful insights to the user as to how the code can be optimized.

As compared to the other existing tools, this thesis will focus on the latest stable version of Solidity (0.8) and will be built around a set principles focused on scalable, maintainable code analysis tools. Security will also not be the main concern, as there is already plenty of interest in that area.

Benchmarking

Unit testing * chestii ciudate gen –yul-optimizations 'r' 'u' 'ru' 'ur' 'rr' 'uu' 'rru' 'uur' 'rur' 'uru' 'uurruurr' * ../test//libyul/yulOptimizerTests * ../test//cmdlineTests/

* test/tools/isoltest –test "yul*/*" ¡- asta!!! * Yul Optimizer Test Summary: 549/564 tests successful (15 tests skipped). * De ce am luat contracte de pe etherscan si nu dintr-un dataset gen https://www.kaggle.com/datasets/xblock/smart-contract-attribute-dataset?reso
* Pentru ca alea din kaggle is deprecated deja, prea multe breaking changes

When benchmarking: ¿ DO NOT benchmark with the solc binary in prerelease mode ¿ ONLY benchmark on contracts WITH A SINGLE function ¿ Reason: dispatch gas consumption. Minute 16:00 de aici: https://www.youtube.com/watch?v=BWO7ij9sLuAlist=
$SoliditySummit$

¿ De vorbit despre mediul de testare ¿ repository forked ¿ link catre schimbarile facute? :-?

¿ Comparatiile de gas – DE LA COMMIT-UL VERSIUNII!! ¿ 80d49f37028b13e162951b6b67b0
¡- commit-ul la care s-a facut release pt v0.8.14 ¿ commit-urile mele ar trb sa vina peste asta

Truffle commands to compile, migrate, and estimate gas consumption

Raspunsul lui chriseth cand primeam eroarea asta:

"' Warning: This is a pre-release compiler version, please do not use it in production.

Error: Source file requires different compiler version (current compiler is 0.8.14-develop.2022.6.1+commit.80d49f37.Darwin.appleclang) - note that nightly builds are considered to be strictly less than the released version –¿ /Users/sergiuiacob/solidity-optimization-with-control-flow-graphs/dataset/termination2.sol:2:1: — 2 — pragma solidity $^0.8.14;$ |

"'

you need to tell the c++ compiler that you actually want to build a release build.

We are working on improving that situation. You do this by doing echo -n ¿ prerelease.txt in the project root

Conclusions

Solidity is still a place where ”construction is in progress”. What’s common across optimizing all of the versions of this programming language is common software techniques, known as static analysis. Control Flow Graphs are a center piece of this mechanism and represents a good starting point, even more valuable when augmented by ASTs and other intermediate representations of the code, such as op code or YUL.

State of the art in Smart Contract Optimization is still receiving inputs, modifications, enhancements from many parts as parties, as the programming language itself is still under heavy development. https://www.overleaf.com/project/627ab6f7ebec3330916a293b

Bibliography

[1] Solidity Official documentation, Solidity Team `https://docs.soliditylang.org/en/v0.8.13/internals/optimizer.html#optimizer-steps`

[2] Solidity Instrumentation Framework (SIF), Chao Peng, Sefa Akca, Ajitha Rajan, University of Edinburgh, 2019 `https://arxiv.org/pdf/1905.01659.pdf`