SIMPLE — Untyped

Grigore Roşu and Traian-Florin Şerbănuţă (grosu,tfserban@illinois.edu)
University of Illinois at Urbana-Champaign

Abstract

SIMPLE is a new, functional, as-fast-as-possible EVM language. SIMPLE is intended for scalability and formal verification. SIMPLE is a low-level language with a very simple syntax, which makes it easy to write tools for it, including formal verification tools. Formal verification tools are intended and designed from the ground up to work for SIMPLE, including a sound and complete formal semantics, a deductive program verification system, and a bounded model checker. This document is a tutorial introduction to the SIMPLE language, its formal semantics, and its deductive program verification system. The bounded model checker is not yet implemented.

Contents

Abstract
Contents
Introduction
Syntax
Semantics
Dynamic Semantics
Static Semantics Infrastructure
Contracts
Configuration
Expressions
Variable Declaration
Variable Initialization
Assignment
Conditional Jumps
Unconditional Jumps
Function Declaration
Function Invocation
Return Statement
Return Data
Stack Manipulation
Memory Manipulation
Storage Manipulation
Calldata Manipulation
Log Instruction
Stop Instruction
Revert Instruction
Invalid Instruction
Self Destruct Instruction
Call Instruction
Call Code Instruction
Delegate Call Instruction
Static Call Instruction
Create Instruction
Block Hash Instruction
Block Header Instructions
Gas Instruction
Program Counter Instruction
Call Data Size Instruction
Code Size Instruction
Return Data Size Instruction
Gas Price Instruction
Address Instruction
Balance Instruction
Origin Instruction
Caller Instruction
Call Value Instruction
Extended Arithmetic Instructions
Comments
Blocks
Scoping
Static Semantics
Dynamic Semantics
Conditional Statement
While Loop
Function Definition
Function Call
Assignment
Expression Statement
Block
Contract
Program
Operational Semantics
Deductive Program Verification
Bounded Model Checking
Implementation
Conclusion
Appendix A: Grammar
Appendix B: Formal Semantics
Appendix C: Deductive Program Verification
Appendix D: Bounded Model Checking
Appendix E: Implementation
Appendix F: Conclusion
Bibliography

Introduction

Syntax

The syntax defining the SIMPLE syntax. The language statements descend down from the syntactical point under consideration to the lowest syntactical unit, which is a literal or an identifier.

Semantics

Dynamic Semantics

The dynamic semantics of SIMPLE is given in the form of a rewriting logic theory. Programs are represented as terms, which are rewritten in the EVM state.

Static Semantics Infrastructure

Before we can start talking seriously about a static semantics, we need to define the basic semantic infrastructure, including the syntax of the static semantics, which is a set of sorts and subsorts, and some basic operations on the syntactical sorts. The static semantics infrastructure is also needed for the dynamic semantics, because it defines the syntax of the EVM state, which is a set of tuples with components of various sorts. The static semantics infrastructure is also needed for the deductive program verification system, which is based on the dynamic semantics, and for the bounded model checker, which is based on the deductive program verification system. In this section we also define the syntax of the EVM state, which is a set of tuples with various components, to be used in implementation of Yul and for the verifier.

Contracts

Configuration

The configuration of SIMPLE consists of an top-level cell, T, holding a SIMPLE-code, a global-memory cell, a global-storage cell, a gas cell, a call-stack cell, a call-value cell, a call-data cell, a return-data cell, a return-flag cell, and an output cell. The top-level cell holds the current contract being executed, the current block header, the current transaction sender, the current transaction origin, the current transaction gas price, the current transaction call data, the current transaction call value, the current block hash, the current transaction gas limit, the current transaction call code, the current transaction call code size, the current transaction call code hash, the current transaction call code offset, the current transaction call code length, the current transaction call code offset, the current transaction call code length, the current transaction call code offset, the current transaction call code length