Specification

Zero Knowledge circuit

for continuation-passing interpreter

March 7, 2022

Abstract

Contents

1 Introduction

In this document we describe the implementation of a *Continuation Passing Style* (CPS) interpreter for a language called Lurk, which was designed specifically for Zero Knowledge Proof (ZKP) systems.

We are going to closely follow the approach proposed in the book "Essentials of Programming Languages" [2].

2 Zero Knowledge Proofs

In this section we present some basic concepts that are required in order to understand this specification.

2.1 Basic concepts

The zero knowledge Apia

- **Setup.**
- **Prove.**
- **Verify.**

SNARKs

The trusted setup problem.

The witness and its relation to non-deterministic memory.

Circuits, R1CS and other formats.

2.2 Backends

Groth16 [3]

Nova [4]

Halo2 [1, 5]

3 Lurk

In this section we shortly describe Lurk concepts.

3.1 Overview

We construct an interpreter for a functional language called Lurk.

It is a Turing-complete language.

The interpreter is based on CPS.

We have an environment where we can manage variable bindings. For instance, the environment can be represented by the following list:

$$\{(\text{var}_0, \text{val}_0), \ldots, (\text{var}_n, \text{val}_n)\}.$$

We have expressions that can represent recursive data and operations.

We have continuations, which can be used to manage control flow of programs.

3.2 Examples

3.2.1 Exponentiation

```
(letrec ((exp (lambda (base exponent)
                 (if (= 0 exponent)
                     1
                     (* base (exp base (- exponent 1)))))))
         (current-env))
```

3.2.2 Fibonacci

```
(letrec ((next (lambda (a b n target)
                 (if (eq n target)
                     a
                     (next b
                           (+ a b)
                           (+ 1 n)
                           target))))
         (fib (next 0 1 0)))
        (current-env))
```

3.3 Components

3.3.1 t, nil

3.3.2 if

3.3.3 lambda

3.3.4 let

3.3.5 letrec

3.3.6 quote

3.3.7 atom

3.3.8 cons, car, cdr

3.3.9 Arithmetic operations

3.3.10 Equality test

3.3.11 Current env

3.4 Expressions

3.5 Environment

3.6 Continuation

3.7 Store

4 Circuit

4.1 Overview

4.2 Gadgets

4.2.1 Macros

Boolean
    Equality
    Pick

4.2.2 Constraints

Arithmetic operations
    Utils

4.2.3 Pointer

Tag
: Hash

4.2.4 Data

allocate
: reverse lookup

4.2.5 Multicase

case
: multicase

: optimization

5 Final remarks

6 References

References

[1] Sean Bowe, Jack Grigg, and Daira Hopwood. Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021, 2019. `https://ia.cr/2019/1021`.

[2] Daniel P. Friedman and Mitchell Wand. *Essentials of Programming Languages, 3rd Edition*. The MIT Press, 3 edition, 2008.

[3] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, pages 305–326, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[4] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. Cryptology ePrint Archive, Report 2021/370, 2021. `https://ia.cr/2021/370`.

[5] Zcash team. The halo2 book. Zcash github, 2022. `https://zcash.github.io/halo2/index.html`.