Zero-Knowledge circuit for Lurk language

A Turing-complete functional programming language for ZKP

Chhi'mèd Künzang[1] Jonathan Gross[1] Eduardo Morais[2]

[1]Lurk Lab
Protocol Labs

[2]Protocol Labs

ZKProof 5, November 2022

Agenda

- Lurk introduction
- Gadgets
- Functional commitments
- Next steps
- Final remarks

Introduction to Lurk

- Continuation-passing style
- Expressions, Environment, Continuation
- Recursive data
- Tiny reduction step
- Turing-complete, recursion, higher-order functions

Examples

- Lurk is Lisp:

```
> (+ 2 3)
INFO  lurk::eval > Frame: 0
Expr: (+ 2 3)
Env: NIL
Cont: Outermost

INFO  lurk::eval > Frame: 1
Expr: 2
Env: NIL
Cont: Binop{ operator: Sum, unevaled_args: (3), saved_env: NIL, continuation: Outermost }

INFO  lurk::eval > Frame: 2
Expr: 3
Env: NIL
Cont: Binop2{ operator: Sum, evaled_arg: 2, continuation: Outermost }

INFO  lurk::eval > Frame: 3
Expr: 5
Env: NIL
Cont: Terminal

[3 iterations] => 5
```

Examples

- Cons:

```
> (cons (cons (cons 1 2) (cons 3 4)) (cons (cons 5 6) (cons 7 8)))
[21 iterations] => (((1 . 2) 3 . 4) (5 . 6) 7 . 8)
```

- If:

```
> (if (eq 1 1) 42 43)
[6 iterations] => 42
```

- Let and lambda:

```
> (let ((square (lambda (x) (* x x))))
    (square 5))
[9 iterations] => 25
```

- Letrec:

```
> (letrec ((sum-upto (lambda (n) (if (= n 0)
                                     0
                                     (+ n (sum-upto (- n 1)))))))

    (sum-upto 100))
[1612 iterations] => 5050
```

Examples

- Commitment:

```
> (commit 123)

[2 iterations] => (comm 0x185e06ceae235e35b0b54a0032c97c22cde058f810583b4fb8aedf2f1c7aa7f2)
```

- Open:

```
> (open 0x185e06ceae235e35b0b54a0032c97c22cde058f810583b4fb8aedf2f1c7aa7f2)

[2 iterations] => 123
```

- Hide:

```
> (hide 42 123) ;; hiding commitment

[3 iterations] => (comm 0x3d60a7b796f2e38b606132f5b710ac9da6a01de47a475bc928a16e949a8ec6cc)
```

- Open:

```
> (open (hide 42 123)) [5 iterations] => 123
```

- Secret:

```
> (secret (hide 42 123)) [5 iterations] => 42
```

LURK

Examples

- Create functional commitment:

```
> (commit (lambda (x) (+ 7 (* x x))))
[2 iterations] => (comm 0x0e9e25aef3319089909c15ebf0e73caf2e342effc984d057223cf21b1194c577)
```

- Open at 9:

```
> ((open 0x0e9e25aef3319089909c15ebf0e73caf2e342effc984d057223cf21b1194c577) 9)
[12 iterations] => 88
```

- Open at 11:

```
> ((open 0x0e9e25aef3319089909c15ebf0e73caf2e342effc984d057223cf21b1194c577) 11)
[12 iterations] => 128
```

Gadgets

- Boolean: and, or, not, xor, etc
- Arithmetic: $+, -, \times, /$
- Comparisons: $>, \geq, <, \leq$
- U64: div/mod operations
- Pointers
- Conditional
- Case and Multicase
- Functional commitments

Pointer

- Tag and Hash
- The preimage of the Hash is the content of the pointer

Figure: Hash preimage

Figure: Allocated pointer

Case

- **Case:** set of clauses indexed by a field element (no repeated index allowed).
- Each **clause** is given by $(\mathbb{F}, (\mathbb{F}, \mathbb{F}))$.
- Finally, a pair $(\mathbb{F}, \mathbb{F})$ defines the default result in case no match is found.
- Example:
  $\{(7, (101, 102)), (12, (221, 222)), (42, (333, 334)), (123, 123)\}$.

Multicase

- **Multicase:** a set of cases sharing the same index ordering.
- We compute a selector only once, and use it for all cases.
- This strategy allows to reduce the number of constraints when compared to repeating the same *cases* separately.
- Example:
  1. $\{(7,(101,102)),(12,(221,222)),(42,(333,334)),(123,123)\}$
  2. $\{(7,(401,402)),(12,(551,552)),(42,(651,652)),(123,123)\}$
  3. $\ldots$

Functional commitment

- 3-ary hash: secret, tag, value
- Pointers to cont and env
- Function privacy. In general this is not easy, because it is necessary to prove the relation is indeed a function. Since Lurk is deterministic, we get this condition basically for free.

Reduction Step

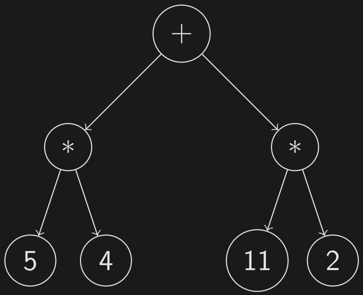

Figure: Expression

$\boxed{\text{outermost}}$

Figure: Continuation

Reduction Step

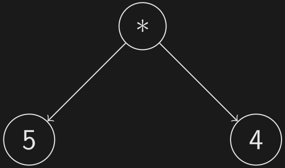

Figure: Expression

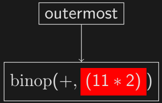

Figure: Continuation

expr means unevaled argument

Reduction Step

Figure: Expression

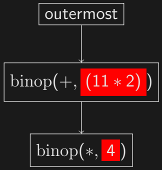

Figure: Continuation

Reduction Step

4

Figure: Expression

$$
\begin{tikzcd}
\boxed{\text{outermost}} \arrow[d] \\
\boxed{\mathrm{binop}(+, \boxed{(11 * 2)})} \arrow[d] \\
\boxed{\mathrm{binop}_2(*, 5)}
\end{tikzcd}
$$

Figure: Continuation

Reduction Step

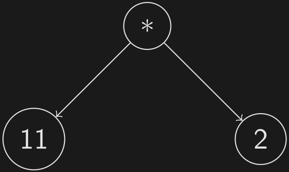

Figure: Expression

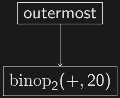

Figure: Continuation

Reduction Step

11

Figure: Expression

$$\begin{array}{c} \boxed{\text{outermost}} \\ \downarrow \\ \boxed{\mathrm{binop}_2(+, 20)} \\ \downarrow \\ \boxed{\mathrm{binop}(*, \boxed{2})} \end{array}$$

Figure: Continuation

Reduction Step

Figure: Expression

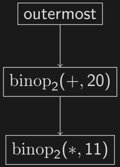

Figure: Continuation

Reduction Step

22

Figure: Expression

$$\begin{array}{c}\boxed{\text{outermost}}\\ \downarrow \\ \boxed{\mathrm{binop}_2(+, 20)}\end{array}$$

Figure: Continuation

Reduction Step

42

Figure: Expression

$\boxed{\text{terminal}}$

Figure: Continuation

Reduction Step

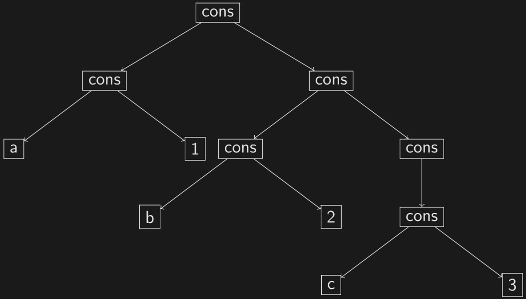

Figure: Lurk environment example

The circuit

- It implements the logic responsible for reducing the expression DAG while correctly managing the environment and the continuation DAGs.
- Reduction step:
  1. Reduce symbol
  2. Reduce cons
  3. Apply continuation
- Each part can be implemented using the multicase gadget, where the clauses are calculated using the other gadgets.

Final remarks

- We are currently implementing U64 and division with remainder.
- New features soon: co-processors, binary-tree IVC, etc.
- Some projects using Lurk: Yatima, Glow, FVM, hardware acceleration.

We are hiring!

- Software Engineer - Applications Development
- Rust Cryptography Engineer
- Documentation Engineer
- Startup Operator and Business Lead

Questions?

Software Engineer - Applications Development

Documentation Engineer

Rust Cryptography Engineer

Startup Operator and Business Lead