Symbolic Computation for fun and for profit

Hari

Devcon Bogota. Oct 13, 2022.

How do we optimize this code?

```
function can_revert(uint x, uint y, uint z) pure returns (uint) {
    if (x < y) {
        if (y < z) {
            if (z < x) {
                revert("bad");
            }
        }
    }
    return 13;
}
```

What is symbolic computation?

- About representing properties using mathematical equations.
- Using solutions of the equations to reason about properties.
  - Usually the system having a solution means a property can be violated.
  - Usually the system having no solutions means a property is always true.

How do we represent a Yul variable?

- Variables in EVM are 256 bit integers.
- Most of the time, you represent variables as an element of integers ($\mathbb{Z}$).
  - If possible, add constraints $0 \leq x \leq 2^{256} - 1$.

How do we assign variables a value?

```
{
    let x := 1
    let y := calldataload(0)
    let z := lt(x, y)
}
```

- We want to represent each assignment by constraints.
- Can we handle every assignment?

```
{
  let x := 1
  switch calldataload(0)
  case 1 { x := 2 }
  case 2 { x := 3 }
  default { x := 4 }
}
```

SSA (Single Static Assignment) Variables

```
{
    let x := calldataload(0)
    let y := calldataload(32)
    // y is not SSA
    y := add(y, calldataload(64))
}
```

But you can transform it into:

```
{
    let x := calldataload(0)
    let y := calldataload(32)
    let z := add(y, calldataload(64))
    // replace all references to y after this by z.
}
```

SSA Variables

- We only want to work with SSA variables.
- It's not always possible to do a Yul to Yul transform such that all variables are SSA.
- But we can still get a lot done. The Yul optimizer has an SSATransform step that transforms Yul into "pseudo SSA format".
- Whenever an non-SSA variable is encountered during analysis, replace it by a "free variable".
  - Each read would be replaced by a fresh free variable.

Encoding EVM Instructions

```
function add(uint x, uint y) pure returns (uint z) {
    z = x + y;
}
```

- For $0 \leq x, y, z \leq 2^{256} - 1$ and $x, y, z \in \mathbb{Z}$.
- Symbolically represent: $z = x + y$?

Add

- EVM semantics: $\text{add}(x, y) = x + y \pmod{2^{256}}$
- $z = x + y \pmod{2^{256}}$.
- Checked arithmetic: the value is only defined when $x + y < 2^{256}$

Let's build a symbolic solver for `lt`, `gt`, `iszero`

$$\text{lt}(a, b) = \begin{cases} 1 & \text{if } a < b \\ 0 & \text{if } b \leq a \end{cases}$$

$$\text{gt}(a, b) = \begin{cases} 0 & \text{if } a \leq b \\ 1 & \text{if } b < a \end{cases}$$

$$\text{iszero}(a) = \begin{cases} 1 & \text{if } a = 0 \\ 0 & \text{otherwise} \end{cases}$$

Difference Logic

- Variables $x_1, \cdots, x_n$ that are integers.
- Constraints of the form $x_i - x_j \leq k_{i,j}$ where $k_{i,j}$ is an constant.

Example:

Let x, y and z be integer variables and let there be constraints:

1. $x - y \leq 4$
2. $x - z \leq 3$

Does the system have a solution?

DL Example

The assignments $x = 4$, $y = 0$ and $z = 1$ satisfies $x - y \leq 4$ and $x - z \leq 3$.

DL Example

What about:

1. $x - y \leq 4$
2. $y - z \leq 3$
3. $z - x \leq -8$

Does this system have a solution?

DL Example

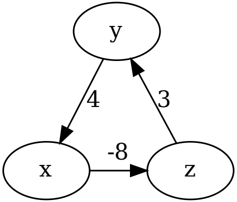
It doesn't have a solution!
Proof: Assume there is a solution, let's add all the three equations:

$$\begin{aligned}(x-y)+(y-z)+(z-x) \leq &\quad 4+3+-8 \\ 0 \leq &\quad -1\end{aligned}$$

Which is a contradiction.

Solver for DL

For a constraint $a - b \leq k$, create nodes a and b with a directed edge from b to a of weight k.
Does it have a negative cycle?



Negative cycles $\iff$ the constraints have no solutions.

Bellman Ford

- Solving DL for unsatisfiablity: look for negative cycle.
- Bellman Ford can be used to compute this.
- Very easy to implement: can even be written in Solidity. See Leo's dl-symb-exec-sol.
- See "Building an End-to-End EVM Symbolic Execution Engine in Solidity" tomorrow at 11:00 for more details.

Insight about unsatisfiablity

- Unsatisfiablity: when the set of constraints have no solution.
- We are generous about ignoring constraints that we can't solve.
- As long as we only care about unsatisfiablity, we can do this.
  - Only optimize when the constraints are unsatisfiable. Otherwise, leave the code unchanged.

`lt`, `gt`, `iszero` as DL constraints[1]

$$\text{lt}(a, b) = \begin{cases} 1 & \text{iff } a - b \leq -1 \\ 0 & \text{iff } b - a \leq 0 \end{cases}$$

$$\text{gt}(a, b) = \begin{cases} 0 & \text{iff } a - b \leq 0 \\ 1 & \text{iff } b - a \leq -1 \end{cases}$$

$$\text{iszero}(a) = \begin{cases} 1 & \text{iff } a - \text{zero} \leq 0 \\ 0 & \text{iff zero} - a \leq -1 \end{cases}$$

In the last example, zero is just a variable we use to indicate zero.

[1] iff: if and only if.

Encoding Yul

- We want to know if the value of an expression is always 0 or always non-zero.
- `if cond { ... }`.
  - Can we replace **`cond`** by **`0`** or **`1`**?
  - Inside the branch, we can add the additional constraint that **`cond = true`**.
- Example: `if lt(x, y) { ... }`
  - Check if adding the constraint $x < y$ makes the system unsatisfiable:
    - In DL: $x - y \leq -1$.
    - replace `lt(x, y)` by `0`.
  - Check if adding the constraint $x \geq y$ makes the system unsatisfiable:
    - In DL: $y - x \leq 0$.
    - replace `lt(x, y)` by `1`.
  - Inside the **`if`** body, add the constraint **`x < y`**.
    - In DL: $x - y \leq -1$.

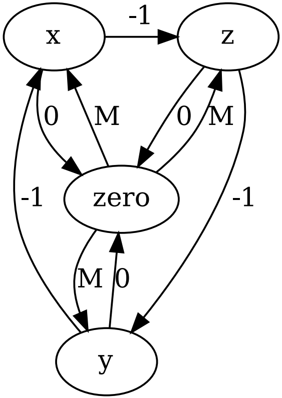
Can this function ever revert?

```
{
    let x := calldataload(0)
    let y := calldataload(32)
    let z := calldataload(64)
    if lt(x, y) {
        if lt(y, z) {
            // should be replaced by `if 0`
            if lt(z, x) {
                revert(0, 0)
            }
        }
    }
}
```

Encoding

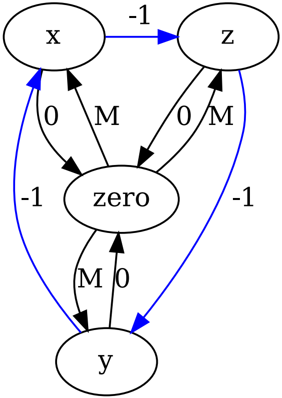
- Define variables $x, y, z \in \mathbb{Z}$.
- No additional constraints from `calldataload(...)`.
- Dummy variable $\text{zero} \in \mathbb{Z}$.
- Add constraints for 256-bit numbers ($0 \leq a \leq 2^{256} - 1$):
  1. $\text{zero} - x \leq 0$, $\text{zero} - y \leq 0$, $\text{zero} - z \leq 0$
  2. $x - \text{zero} \leq 2^{256} - 1$, $y - \text{zero} \leq 2^{256} - 1$, $z - \text{zero} \leq 2^{256} - 1$
- Inside each `if` branch, add the corresponding `lt` constraints:
  1. $x - y \leq -1$
  2. $y - z \leq -1$
  3. $z - x \leq -1$

Graph of the encoding[2]



[2] $M = 2^{256} - 1$.

Negative cycle? Unsatisfiable?[3]



[3] $M = 2^{256} - 1$.

Can this function ever revert?

```
{
    let x := calldataload(0)
    let y := calldataload(32)
    let z := calldataload(64)
    if lt(x, y) {
        if lt(y, z) {
            // Replace `if lt(z, x)` by `if 0`
            if 0 {
                revert(0, 0)
            }
        }
    }
}
```

Proofs

- If we don't trust the solver, we can ask it to produce a proof.
- The proof in this case would be a set of constraints whose LHS would add up to 0 and RHS to negative.
  - This can be verified.

Statically analysing reachability and inferring constraints

```
error OutOfBounds();
contract C {
    uint[] arr;
    function f(uint idx) external view returns (uint) {
        if (idx >= arr.length) revert OutOfBounds();
        // compiler auto generates, the bound checks here.
        // But we can infer the constraint `idx < arr.length`
        return arr[idx];
    }
}
```

- Try to see if a branch will always terminate: either by reverting or returning.
    - Add the opposite constraints outside the branch.

Improvements

- Difference logic only allowed constraints of the form $x - y \leq k$.
- Next step: constraints of the form:

$$a_1 \cdot x_1 + a_2 \cdot x_2 + \cdots + a_n \cdot x_n \leq b$$

  - where a_i and b are constants and x_i is a symbolic variable in integers[4] for $i = 1, \cdots, n$.
- Linear programs and the Simplex method.
- You can encode `add` and `sub`.
  - Requires branching to handle wrapped arithmetic.
- Encode `mul(x, a)` and `div(x, a)` where a is a constant and x is symbolic.

[4] We'll have to relax to Rational or Reals for faster solvers.

Slides

https://hrkrshnn.com/t/devcon-bogota.pdf