⊕	RawBal	NetBal	AccBal	T-Supply	Fee	Debt	Interest	Dividend
RawBal	RawBal	✗	✗	T-Supply	✗	✗	✗	AccBal
NetBal	✗	NetBal	✗	T-Supply	✗	✗	✗	AccBal
AccBal	✗	✗	AccBal	✗	✗	✗	✗	✗
T-Supply	T-Supply	T-Supply	✗	T-Supply	T-Supply	✗	✗	✗
Fee	✗	✗	✗	T-Supply	Fee	Debt	Interest	✗
Debt	✗	✗	✗	✗	Debt	Debt	Debt	✗
Interest	✗	✗	✗	✗	Interest	Debt	Interest	✗
Dividend	AccBal	AccBal	✗	✗	✗	✗	✗	Dividend

Table 1: Definition of Operators ⊕ (top_row ⊕ left_column)

⊗,⊘	RawBal	NetBal	AccBal	T-Supply	Fee	Debt	Interest	Price	Dividend
RawBal	RawBal	✗	✗	RawBal	✗	✗	✗	RawBal	✗
NetBal	✗	NetBal	✗	✗	✗	✗	✗	NetBal	✗
AccBal	✗	✗	AccBal	✗	✗	✗	✗	AccBal	✗
T-Supply	RawBal	✗	✗	✗	✗	✗	✗	T-Supply	✗
Fee	✗	✗	✗	✗	✗	✗	✗	Fee	✗
Debt	✗	✗	✗	✗	✗	✗	✗	Debt	✗
Interest	✗	✗	✗	✗	✗	✗	✗	Interest	✗
Price	RawBal	NetBal	AccBal	T-Supply	Fee	Debt	Interest	Price	Dividend
Dividend	✗	✗	✗	✗	✗	✗	✗	Dividend	✗

Table 2: Definition of Operator ⊗ (top_row ⊗ left_column)

⊗,⊘	RawBal	NetBal	AccBal	T-Supply	Fee	Debt	Interest	Price	Dividend
RawBal	Price	✗	✗	Price	✗	✗	✗	✗	✗
NetBal	✗	Price	✗	✗	✗	✗	✗	✗	✗
AccBal	✗	✗	Price	✗	✗	✗	✗	✗	✗
T-Supply	Price	✗	✗	✗	✗	✗	✗	✗	✗
Fee	✗	✗	✗	✗	✗	✗	✗	✗	✗
Debt	✗	✗	✗	✗	✗	✗	✗	✗	✗
Interest	✗	✗	✗	✗	✗	✗	✗	✗	✗
Price	RawBal	NetBal	AccBal	T-Supply	Fee	Debt	Interest	Price	Dividend
Dividend	✗	✗	✗	✗	✗	✗	✗	✗	✗

Table 3: Definition of Operator ⊘ (top_row ⊘ left_column)

SUPPLEMENTARY MATERIAL

1 DEFINITIONS OF OPERATORS ⊕, ⊗, AND ⊘

Table 1 shows the definition for ⊕. The cells in this table represent the results of the various additions between financial types. For example, `Debt` ⊕ `Fee` = `Debt` (column 7, row 6); `Fee` cannot be added to various types of balances `RawBal`, `NetBal`, or `AccBal` as fee is a charge (see the `deposit()` and `withdraw()` functions in Figure 3); `AccBal` cannot be added to `T-Supply` as accrued balance includes dividend that is taken from total supply and hence adding it back is problematic; `Debt` is not allowed to be added to any sort of balances which denote the assets a user owns.

Table 3 shows the definition for ⊗. For example, `NetBal` ⊗ `Price` = `NetBal` (column 3, row 9). Intuitively, a net balance that is converted to another currency by multiplying a price is still a net balance. `Fee`, `Debt`, `Interest`, and `Dividend` are only allowed to have multiplication with `Prince`, but not any other types. The same kind of balances are allowed to multiply (e.g., `RawBal` ⊗ `RawBal`). This happens in computing price curves. But multiplication of different kinds of balances are disallowed due to their incompatible nature.

Table 3 shows the definition for ⊘. Divisions of balances of the same type yield `Price`. Any type divided by a `Price` yields the same type. This often happens during trading. `Price` divided by `Price` corresponds to getting price through a chain of trading.

2 SCTYPE ARTIFACT AND BENCHMARK

SCType, our tool, is a Solidity type checker written in Python 3. There are two Docker Images that we have provided for our tool. Both of the images contain the tool, as well as the benchmarks we use. They can be found at an anonymous site [79].

The full Docker Image holds the complete benchmark set and all of their dependencies. It includes 29 projects. It can be downloaded by running

```
>docker pull icse24sctype/full
```

It is 23 GB as all the dependences of all these projects have to be in place for Solidity and Slither.

The reduced Docker Image holds one benchmark case and its dependencies. It can be downloaded by running

```
>docker pull icse24sctype/min
```

It is 1 GB.

Benchmark. Our benchmark test cases can be found in the 'Benchmark' directory. The reduced version only contains one benchmark case, which is `Vader Protocol p1`. The smart contract tested is the `Utils.sol` file in the `Benchmark/Vader_Protocol_p1/vader-protocol/contracts` directory. The full version contains all of the benchmarks.

Tests. To run the reduced tests, run the following command in a Linux environment:

```
>./test_minbenchmark.sh
```

To run the full tests, run the following command:

```
>./test_benchmark.sh'
```

The type-checking results are in green. The output of the warnings is: Variable name (IR), Function name, and Operation with mistake. Please ignore all other outputs. One of the three warnings describing the function 'calcLiquidityUnits()' of the second benchmark in the full version or the only benchmark in the reduced version is the example case we explain in our Motivation section.

Source The source code for our tool can be found in the directory: `slither/detectors/my_detectors`. In particular, `tcheck.py` is the main engine.