Contract Invariants

Properties that should always hold. Contract invariants are properties that we expect to be maintained by the contract at 'all times'.

Invariant example: "The total of all balances doesn't change"

Contract invariants in Scribble are similar to class invariants in object oriented languages - internal functions to the contract are allowed to temporarily violate the invariants. However whenever we"exit" the contract (by either returning from a call into it, or by making an external call from it), the invariantsmust be true . We refer to these points in the execution as places where the state is externally observable.

Observable State

The state of a contract is observable whenever some external contract can read it.

before call (observable) : The state of a contract before executing a transaction is observable! At this point anyone, and any contract could read and call into a contract. Those callers / readers expect the invariant to hold.

during call (not-observable) : During a contract's execution no external cotnract looks at the state of the contract, since no other contract is executing. So it's fine if an invariant doesn't hold at this point.

before making external call out (observable) : If we call to another contract, then that contract can depend our state or call back. This external contract expects the invariant to hold!

after (returning from) external call (observable) : Once a contract is finished executing we are back to a scenario where anyone and any contract can depend on our state. So, the invariant should hold again at this point.

Example Observability

Let's explore this with an example. The invariant we'll use is "the variable x is 0"

```

Copy /// #invariant {:msg "the variable x is 0"} x == 0; contract Example { // Good, x holds in the beginning uint public x = 0;

function example_observability() { // x should be 0 here x *= 10; // x is still 0 if x was zero return; // x is still zero if x was zero, great! }

function example_violation() { // x should be 0 here x += 1; return; // x is not zero anymore ☹ // now the invariant is violated }

function example_nonviolation() { // x should be 0 here x += 1; // Now x is non zero, but that doesn't matter. We're not "observable". x -= 1; // We immediately return x back to what it was return; // x is back to zero again (if it was zero before)! }

function example_external(address externalContract) { // x should be 0 here x += 1; // Now x is non zero. Which violates the invariant. externalContract.call();

// externalContract might call back into us, assuming that the invariant holds // For example: // If externalContract calls this.example_observability(), expecting all invariants to hold. // Unfortunately, x is not 0. example_observability()'s assumption is violated.

return; }

}

```

Syntax

Invariants have the following structure, and are placed above contract declarations:

```

Copy invariant {:msg '"'} ;

```

Example 1 Example 2 This is a property could be used in a token contract to assert that the token is not inflationary or deflationary.

```
```

Copy /// #invariant {:msg "The total balance doesn't change."} totalBalance == old(totalBalance); contract ExampleContract {

}

``` The following property is very common. It checks that if the owner of a contract changes, then msg.sender must have been the old owner.

```

Copy /// #invariant "Only the owner changes the owner" owner != old(owner) ==> msg.sender == old(owner); contract ExampleContract { address public owner; }

```

External Calls

Don't use external calls in invariants! There are two kinds of external calls: Ones that call tothis and ones that call into other contracts.

Scribble asserts that an invariant is valid at the end of contract construction, as this is the first point where contract state becomesobservable . However, due to the mechanics of the EVM the contract account is not 100% set up. Calls into the contract don't work yet, because there is not actually any code at the contract address until the constructor returns. This is why you can't use re-entrant calls likethis.getTotalBalance() in a contract invariant.

The second type of external call, those to other contracts, is possible with Scribble. But we consider this an anti-pattern.

Invariants are meant to describe properties that must always hold fora single contract. We write properties for the contractin isolation, agnostic of any external components. This makes testing more efficient and reflects the composability of (DeFi) smart contracts.

Previous State Variable Annotations Next Scribble Functions Last updated2 years ago

On this page * Observable State * Example Observability * Syntax * External Calls

Was this helpful?
```