Introduction

All scribble properties are expressed as specially formatted docstrings. For example:

```

Copy contract Foo { /// #if_succeeds {:msg "P0"} y == x + 1; function foo(uint256 x) public returns (uint256 y) { return x + 1; } }

```

In the above sampleif_succeeds {:msg "P0"} y == x + 1; is the property. Note thesemicolon at the end - it is required for all properties.

There are several kinds of Scribble annotations.

Property Annotations

Property annotations directly correspond to specific properties the user wants to check. There are 4 types of properties supported currently:

1. if_succeeds
2. properties are usually specified before functions. They specify conditions that must hold if the function returns successfully. (if_succeeds
3. can also be specified on contracts. For more details on their semantics seethis
4. section).
5. invariant
6. properties are specified only before contracts. They specify conditions that must hold whenever we are NOT executing code in the contract (more info inthis
7. section).
8. if_updated
9. properties are specified only before state variables. They specify conditions that must be checked whenever we are updating the value of a state variable (or some of part of the state variable in the case of complex data structures such as arrays, structs and maps). More info inthis
10. section.
11. assert
12. properties are specified before a statement in the body of the function. They specify a condition that must be checked inline before the target statement, and can refer to any local variable in scope for the target statement. For more details seethis
13. section.
14.

All properties allow adding a human-readable message using{:msg "message"} . This is helpful when mapping the properties to their description in other documents, or just as a reminder about the intent of a complex property.

Since its tedious to write{:msg "some message"} you can also just write "some message" as shorthand. So for example the below is a valid annotation:

/// #if_succeeds "x increases" x > old(x); The expressions that are allowed in custom properties are mostly a subset of the pure Solidity expressions, with some additional syntactic sugar. For more details seethis section.

Utility Annotations

Utility annotations are helpers that make writing property annotations easier. We currently support:

1. Scribble user-defined functions
2.
    - helpful for defining commonly used predicates and reducing code duplication
3. Scribble macro annotations
4. allow automatically annotating a contract with a whole set of predefined annotations (from a macro definition)
5. Backend tool hints
6. (#try
7. and#require
8.) - annotations providing guidance to backend tools
9.

Examples

To give you a quick intro to scribble annotations lets look at two small examples uinsg the#if_succeeds annotation.

Using old

In the below contract the functioninc increments the value of the state variablex by 1. The propertyx == old(x) + 1 captures exactly this behavior.

```

Copy contract Foo { int x = 1; /// #if_succeeds {:msg "P0"} x == old(x) + 1; function inc() public { x = x + 1; } }

```

Note that we can also write that property asx == old(x + 1) since the value of the constant1 doesn't change before and after the function.

Incrementing contract

For the contract from the previous example we can also add a contract invariant stating thatx >= 1 as shown below:

```

Copy /// #invariant {:msg "P1"} x >= 1; contract Foo { int x = 1; /// #if_succeeds {:msg "P0"} x == old(x) + 1; function inc() public { x = x + 1; } }

```

Expressing pre/post conditions using implication

In verification people often think of function pre- and post- conditions (these come fromHoare logic). One can encode a preconditionP and post-conditionQ as:P ==> Q . So for example for thesillyDiv function below, we can express a pre-condition (y != 0) and post-condition (z == x/y) asy!=0 ==> z == x/y .

```

Copy contract Div { /// #if_succeeds {:msg "P0"} y != 0 ==> z == x / y; function sillyDiv(uint x, uint y) public returns (uint z) { return y != 0 ? x/y : 0; } }

```

Note that this specification is incomplete - it doesn't say what happens wheny == 0 . We can make it complete by adding another property:

```

Copy contract Div { /// #if_succeeds {:msg "P0"} y != 0 ==> z == x / y; /// #if_succeeds {:msg "P0"} y == 0 ==> z == 0; function sillyDiv(uint x, uint y) public returns (uint z) { return y != 0 ? x/y : 0; } }

```

Multipleif_succeeds annotations for the same functions areconjoined together. The same holds for multipleinvariant annotations for a contract.

Previous FAQ Next Annotations Last updated2 years ago