Function Annotations

Function specifications are used to specify the behaviour of a single function.

Successful termination

Scribble currently has a single type of function specification, calledif_succeeds . Using if_succeeds, annotations you can assert properties that you expect to hold if (and only if) the annotated function successfully terminates.

These annotations have the following structure & are placed right above a function declaration:

```

Copy if_succeeds {:msg ""} ;

```

Example 1 The following is a simple example property, where we check that the function always returns 0.

```

Copy contract ExampleContract { /// #if_succeeds {:msg "result is always zero"} result == 0; function example() public returns (uint result) { ... } }

``` Note that as of Scribble 0.4.0 you can also specify anif_succeeds annotation at the contract level. See thesection below .

Encoding Pre- and Post- Conditions

Pre- and postconditions are frequently used concepts in verification languages.

1. Aprecondition
2. is what should be true before a function is executed.
3. Apostcondition
4. is what should be true after the function,if
5. the precondition was true.
6.

In it's simplest formif_succeeds allows you to write postconditions (i.e. properties over the state after the function is done).

However, if_succeeds is quite versatile and you can use it to check preconditions as well. You'd do so by using theold() function .

Example 1 Example 2 We expect that a selfdestruct function only succeeds if the transaction sender is the owner of the contract.

```

Copy /// #if_succeeds {:msg ""} old(msg.sender == owner); function selfdestruct() public onlyOwner {}

``` We expect that the following function will only succeed if the input variableamount is not zero.

```

Copy /// if_succeeds {:msg "Amount is not zero"} old(amount) != 0; function transfer(uint amount, addres receiver) public { ... }

```

We're using old( ... ) here to make sure that the property is checked with the value ofamount at the beginning of the transaction. This is because the value ofamount can change during the transaction!

Encoding conditional "behaviors"

Functions can have multiplebehaviors depending on the inputs and contract state. Take the following example of a simple calculator function implementing multiplication and addition.

```

Copy contract Calculator { enum Op { PLUS, TIMES }

function calc(Op operator, uint left, uint right) public returns (uint result) { if (operator == Op.TIMES) { return left * right; } if
```

(operator == Op.PLUS) { return left + right; } assert (false); } }

```

In this case, there are two distinct behaviors:multiplication andaddition . Writing two separate post-conditions wouldn't work, sincescribble checks theconjunction of all function annotations.

```

Copy /// #if_succeeds "result is product of left and right" result == left * right; /// #if_succeeds "result is sum of left and right" result == left + right; function calc(Op operator, uint left, uint right) public returns (uint result) { ... }

```

Luckily, you can use the implication operator here to describe the conditions under which each behavior holds:

```

Copy /// #if_succeeds "result of times operator is product of left and right" operator == Op.TIMES ==> result == left * right; /// #if_succeeds "result of plus operator is sum of left and right" operator == Op.PLUS ==> result == left + right; function calc(Op operator, uint left, uint right) public returns (uint result) { ... }

```

Contract-level if_succeeds annotations

As of Scribble 0.4.0 you can also specify anif_succeeds annotation at the contract level. A contract-levelif_succeeds is automatically applied toall public and external functions in the annotated contract, and all inheriting contracts. For example, in the below sample the annotationx > 0 will be applied to the functionsA.pubFun ,A.externFun andB.pubFunB . Note thatpure andview functions are skipped. So in the below exampleA.pubViewFun will not be annotated.

```

Copy /// #if_succeeds x > 0; contract A { uint x; function pubFun() public { ... } function externFun() external { ... } function internalFun() internal { ... } function pubViewFun() external view { ... } }

contract B is A { function pubFunB() public { ... } function privFunB() private { ... } }

```

Note that since the same expression needs to hold for an unknown set of functions, when writing a contract-levelif_succeeds function you can only talk about the contract's state vars, and other global transaction state. You cannot refer to function arguments/returns (even if all functions in the contract have the same arguments or returns).

Previous Annotations Next State Variable Annotations Last updated2 years ago
```