Assert Annotations

Checking properties in the middle of a function Most properties discussed so far can only reference the contract's state variables (invariant andif_updated) or the contract's state variables and the arguments/returns of a specific function (if_succeeds). While this is already powerful, sometimes you really want to check a property in the middle of an existing function (e.g. inside a loop?) and wish to talk about the values of local variables in the function. For this purpose we have added the#assert annotation.

Motivating Example

Consider the following code sample:

```

Copy contract Administrable { mapping(address => boolean) admins; function setAdmins(address[] memory newAdmins) public { for (uint i = 0; i < newAdmins.length; i++) { admins[newAdmins[i]] = true; } } }

```

One property we might want to express is that"the zero account is never made an admin". We can already express this using the universal quantifierforall in the following way:

```

Copy /// #if_succeeds forall(uint i in newAdmins) newAdmins[i] != 0; function setAdmins(address[] memory newAdmins) public {...

```

Or even with anif_updated annotation like so:

```

Copy /// #if_updated forall(uint i in admins) admins[i] != 0; mapping(address => boolean) admins;

```

However in both those cases, the instrumentation will have to re-writeadmins state variable into a custom type that tracks the set of keys. This adds multiple arrays and maps, and also adds extra loops in the instrumentation code (implementing the actualforall checks). This instrumentation can sometimes cause trouble for backend tools (such as fuzzers and symbolic execution engines) because of all of the extra paths its adding to the code.

An alternative, more light-weight annotation you might want to add is just a check in the loop in the body ofsetAdmins that makes sure that each assigned address is not 0. You can express this like so:

```

Copy function setAdmins(address[] memory newAdmins) public { for (uint i = 0; i < newAdmins.length; i++) { /// #assert "zero is not an admin" newAdmins[i] != 0; admins[newAdmins[i]] = true; } }

```

The above annotation works exactly as a normalassert() would inserted in this case.

Syntactic Rules about #assert annotations

There are several things to remember about#assert :

1.#assert can only be inserted directly before another statement, and in the instrumented code they would also be inserted before this statement.

You cannot insert an

assert

annotation at the end of a code block! If you do, it will be silently ignored! This limitation is due to the way the solidity compiler reports docstrings on statements to us. We are working on resolving this. Until then, if you need to insert an annotation at the end of a code block, insert a dummy no-op statement. For example:

... /// #assert false; 0; // Dummy no-op statement inserted

} 2. #assert can talk about any of the local variables that are in scope at the point where it is defined.

1. You cannot useold() inside of anassert()
2. it can only talk about the state of the program at the point of the#assert . If you want to talk about changes in state across some statements, check out the articles on the new experimental let feature(TODO).

Previous Scribble Functions Next Macros Last updated2 years ago