Macros

Annotations for standardized smart contracts There are several classes of standardized smart contracts - e.gERC20 ,ERC721 ,ERC777 . While there are variations in their implementation, there are a lot of common properties you would expect to hold across implementations.

To avoid manually copying and pasting the same properties for each implementation, in Scribble0.6.1 we added contract-level macros. A macro consists of 2 parts:

1. A macro definition - a.yaml file describing the target contract and the properties that the macro should insert
2. A#macro (, ...); annotation on the target contract, that instantiates all the properties from the corresponding.yaml file.

Lets explore this withERC20 as an example.

Example: The ERC20 macro

Lets try to define some common properties for a class ofERC20 tokens. AllERC20 tokens must for example define atransferFrom andtotalSupply functions, and many implementations will have a internal map from addresses to balances. We may want to check several properties of such tokens. For example:

- whentransferFrom(from, to, amount)
- is called, the balance of the sender decreases by amount, and the balance of the receiver increases by amount
- the return value oftotalSupply
- is always equal to the sum of the balances
- No function changes the total supply of the tokens (assuming we are only talking about non-deflationary tokens)
-

Macro Definitions

Here is how we can define a new macro in an yaml file, that checks the above properties when instantiated:

```

Copy erc20: variables: balances: mapping (address => uint256) properties: balances: - msg: "The token has a fixed supply (only set in constructor)." prop: "#if_updated msg.sig == bytes4(0x00000000);" totalSupply(): - msg: "Result is equal to sum of balances" prop: "#if_succeeds result == unchecked_sum(balances);" transferFrom(from, to, amount): - msg: "The sender looses amount" prop: "#if_succeeds from != to ==> old(balances[from]) - amount == balances[from];" - msg: "The receiver receives amount" prop: "#if_succeeds from != to ==> old(balances[to]) + amount == balances[to];"

```

There are several things to unpack here:

1. On line 1 we give the name of the macro. This will be how we will refer to this macro when instantiating it.
2. In thevariables
3. section (lines 2-3) we list the variables that we expect the target contract to have. Note that in the actual contract the variables can have different names! We will see how this renaming happens in the next section
4. In theproperties
5. section, we have a list of "annotation targets" and for each target the list of annotations that must be applied to this target. An annotation target can be a function signature, state variable name, or
6. which applies to the whole contract.
7. Each annotation has two fields -msg:
8. is the human readable label that should be applied to the property andprop:
9. is the actual scribble annotation.
10.

So for example the below fragment of the yaml definition, specifies that scribble should find the functiontotalSupply() in the target contract, and insert#if_succeeds {:msg "Result is equal to sum of balances" result == unchecked_sum(balances); right above it:

```

Copy totalSupply(): - msg: "Result is equal to sum of balances" prop: "#if_succeeds result == unchecked_sum(balances);"

```

Macro Invocations

Lets look at how you apply a macro, and what happens in more detail. In the below example we apply the macro defined in

the previous section to theMyToken contract.

```

Copy /// #macro erc20(_balances); contract MyToken { mapping(address=>uint256) _balances; function totalSupply() public view (returns uint256) { ... }; function transferFrom(address sender, address reciver, uint256 value) public returns (bool) { ... } ... }

```

The macro invocation is done with the#macro erc20(_balances) annotation in the docstring above the target contract. The macro invocation gives the macro name (erc20) and it lists theactual names of all variables required in thevariables section of the macro definition.

In thevariables section of theerc20 macro definition we mentioned abalances variable. Some properties declared in theerc20 macro definition also talk about abalances variable. HoweverMyToken doesn't have abalances state variable - instead it has a state variable named_balances . By specifyingerc20(_balances) in the macro invocation, we are telling scribble to replace all occurances ofbalances in the macro definition with_balances .

Additionally note that the arguments oftransferFrom in the definition are namedfrom ,to andamount . On the other hand inMyToken these same arguments are namedsender ,receiver andvalue . Scribble will automatically rename the function arguments of function properties when instrumenting the contract.

So after the macro is instantiated, the above contract will be annotated as follows:

```

Copy contract MyToken { /// #if_updated {:msg "The token has a fixed supply (only set in constructor)."} msg.sig == bytes4(0x00000000); mapping(address=>uint256) _balances; /// #if_succeeds {:msg "Result is equal to sum of balances"} result == unchecked_sum(_balances); function totalSupply() public view (returns uint256) { ... }; /// #if_succeeds {:msg "The sender looses amount"} from != to ==> old(balances[from]) - amount == balances[from]; /// #if_succeeds {:msg "The receiver receives amount"} from != to ==> old(balances[to]) + amount == balances[to]; function transferFrom(address sender, address reciver, uint256 value) public returns (bool) { ... } ... }

```

Macro Locations

Scribble looks for macros recursively in the "macros" directory. You can specify the macros directory using the--macro-path command line options.

As of Scribblev0.6.3 we also ship a small standard library of macros which is searched in addition to the explicit--macro-path location. The standard macro library is located underSCRIBBLE_INSTALL_PATH/stdlib whereSCRIBBLE_INSTALL_PATH is the location where npm has installed theeth-scribble package. Previous Assert Annotations Next Hints Last updated2 years ago