Quantification

Quantification allows us to talk about all values in a set

Universal Quantification

Syntax:

forall (in ...)

forall (in)

Universal quantifiers are powerful expressions that allow users to talk about all values that fit a criteria. Scribble supports 2 types of universal quantifiers - quantifiers over numeric ranges and quantifiers over arrays and maps.

Numeric ranges

Quantifiers can talk about properties that hold over all numbers in a given range. The syntax for such quantifiers isforall(in ...) .

The start expression isinclusive , while the end range expression isexclusive. The most common use-case of quantification over numeric ranges is describing all indices of an array (see below example).

Example 1: Forall over arrays In the below example we check that the arrayarr is sorted.

Copy /// #invariant forall(uint i in 0...arr.length-1) arr[i] < arr[i+1]; contract Token { uint[] arr; ... }

``` Note that the type of the quantifier variable (uint i in the above example) must always be a numeric type, and it must be able to fit both the start and end expressions. So for example you can't write:forall (int8 i in 0...arr.length-1) ... in the above example instead, sinceint8 is too small of a type to store the array length (which isuint256 ).

Arrays/maps

The second flavor of universal quantifiers talks about all indices in an array or keys in a map. The syntax isforall (in ) .

Example 1: Forall over arrays Example 2: Forall over maps In the below example, we check that each stakeholder has a positive stake

```

Copy /// #invariant /// forall(uint i in stakeHolders) stakes[stakeHolders[i]] > 0; contract Token { address[] stakeHolders; mapping(uint=>uint) stakes; }

``` In the below example, the mappingauthorized is a mapping from account addresses to operators addresses, authorized to act on behalf of the given account. The property below states that for no account is the0x0 address authorized to act on their behalf.

```

Copy /// #invariant /// forall(address acct in authorized) authorized[acct] != address(0); contract Token { mapping(address=>address) authorized; }

``` Anyforall over arrays can be equivalently expressed as aforall over numeric ranges. I.e.forall(i in arr) is equivalent toforall(uint i in 0...arr.length) Note that in Solidity maps are technically defined over their complete input range (i.e. amapping(uint=>uint) is defined for alluint256 s, just its 0 for most of them). Due to this, when we say that we "quantify over all keys in a map", we mean all keys that have:

1. Been set explicitly at least once
2. Have not been deleted with thedelete keyword.

To support this, under the hood we re-write all annotated maps with our custom data type that track key insertion and deletions.

Thedelete operation is equivalent to zeroing out a value. However, we do not treat them equivalently. Given a mapm , when you dodelete m[x] , we will removex from the keys that we store inm . However, if you dom[x] = 0 we willnot remove x from the keys we store inm . You can nestforall statements arbitrarily. For example, we can extend the example above to store a list of authorized users for a given address and require that none of the authorized users is0x0 :

Authorized Users 2 In the below example, the mappingauthorizedUsers stores a list of addresses of operators authorized to
```

act on behalf of a given address. The property below states that the0x0 address is not authorized to act on anyone's behalf.

```
Copy /// #invariant /// forall(address acct in authorized) /// forall(uint i in authorized[acct]) /// authorized[acct][i] != address(0);
contract Token { mapping(address=>address[]) authorized; }
```

Previous Expressions Next Property Checking with Scribble and MythX Last updated2 years ago

On this page * Universal Quantification * Numeric ranges * Arrays/maps

Was this helpful?