Dafny - Sets

Microsoft Research

Translate this page
Chinese Simplified ▼
Microsoft® Translator

other tutorials close

Sets

1. tutorials

Sets of various types form one of the core tools of verification for Dafny. Sets represent an orderless collection of elements, without repetition. Like sequences, sets are immutable value types. This allows them to be used easily in annotations, without involving the heap, as a set cannot be modified once it has been created. A set has the type:

```
set<int>
```

for a set of integers, for example. In general, sets can be of almost any type, including objects. Concrete sets can be specified by using display notation:

load in editor

```
var s1 := {}; // the empty set
var s2 := {1, 2, 3}; // set contains exactly 1, 2, and 3
assert s2 == {1,1,2,3,3,3,3}; // same as before
var s3, s4 := {1,2}, {1,4};
```

The set formed by the display is the expected set, containing just the elements specified. Above we also see that equality is defined for sets. Two sets are equal if they have exactly the same elements. New sets can be created from existing ones using the common set operations:

load in editor

```
assert s2 + s4 == {1,2,3,4}; // set union
assert s2 * s3 == {1,2} && s2 * s4 == {1}; // set intersection
assert s2 - s3 == {3}; // set difference
```

Note that because sets can only contain at most one of each element, the union does not count repeated elements more than once. These operators will result in a finite set if both operands are finite, so they cannot generate an infinite set. Unlike the arithmetic operators, the set operators are always defined. In addition to set forming operators, there are comparison operators with their usual meanings:

load in editor

```
assert {1} <= {1, 2} && {1, 2} <= {1, 2}; // subset
assert {} < {1, 2} && !({1} < {1}); // strict, or proper, subset
assert !({1, 2} <= {1, 4}) && !({1, 4} <= {1, 4}); // no relation
assert {1, 2} == {1, 2} && {1, 3} != {1, 2}; // equality and non-equality
```

Sets, like sequences, support the `in` and `!in` operators, to test element membership. For example:

load in editor

```
assert 5 in {1,3,4,5};
assert 1 in {1,3,4,5};
assert 2 !in {1,3,4,5};
assert forall x :: x !in {};
```

Sets are used in several annotations, including reads and modifies clauses. In this case, they can be sets of a specific object type (like `Nodes` in a linked list), or they can be sets of the generic reference type `object`. Despite its name, this can point to any object or array. This is useful to bundle up all of the locations that a function or method might read or write when they can be different types.

When used in a decreases clause, sets are ordered by subset. This is unlike sequences, which are ordered by length only. In order for sets to be used in decreases clauses, the successive values must be "related" in some sense, which usually implies that they are recursively calculated, or similar. This requirement comes from the fact that there is no way to get the cardinality (size) of a set in Dafny. The size is guaranteed to be some finite natural, but it is inaccessible. You can test if the set is empty by comparing it to the empty set (`s == {}` is true if and only if `s` has no elements.)

A useful way to create sets is using a set comprehension. This defines a new set by including `f(x)` in the set for all `x` of type `T` that satisfy `p(x)`:

```
set x: T | p(x) :: f(x)
```

This defines a set in a manner reminiscent of a universal quantifier (`forall`). As with quanifiers, the type can usually be inferred. In contrast to quantifiers, the bar syntax (`|`) is required to seperate the predicate (`p`) from the bound variable (`x`). The type of the elements of the resulting set is the type of the return value of `f(x)`. The values in the constructed set are the return values of `f(x)`: `x` itself acts only as a bridge between the predicate `p` and the function `f`. It usually has the same type as the resulting set, but it does not need to. As an example:

load in editor

```
  assert (set x | x in {0,1,2} :: x * 1) == {0,1,2};
```

If the function is the identity, then the expression can be written with a particularly nice form:

load in editor

```
  assert (set x | x in {0,1,2,3,4,5} && x < 3) == {0,1,2};
```

General, non-identity functions in set comprehensions confuse Dafny easily. For example, the following is true, but Dafny cannot prove it:

load in editor

```
  assert (set x | x in {0,1,2} :: x + 1) == {1,2,3};
```

This mechanism has the potential to create an infinite set, which is not allowed in Dafny. To prevent this, Dafny employs heuristics in an attempt to prove that that the resulting set will be finite. When creating sets of integers, this can be done by bounding the integers in at least one clause of the predicate (something like `0 <= x < n`). Requiring a bound variable to be in an existing set also works, as in `x in {0,1,2}` from above. This works only when the inclusion part is conjuncted (`&&`'ed) with the rest of the predicate, as it needs to limit the possible values to consider.

tutorials

- Guide
- Termination
- Sets
- Sequences
- Collections
- Lemmas
- Modules