Map

Maps are a collection where the elements are indexed. Elements may be members of the collection (its range) more than once, but key values must be unique. A Map m of type $Map(K, T)$ is considered to be based on an underlying set $m \rightarrow asSet()$ of pairs $Tuple(first : K, second : T)$ where the first element is the key and the second the value. For convenience we write such pairs as maplets $first \mapsto second$, and literal maps as

$$Map\{k1 \mapsto v1, ..., kn \mapsto vn\}$$

Map types occur in UML as the type of qualified associations, or as indexes of objects by a key value. They can be used to implement symbol tables for formally-specified software tools, and to implement operation caching.

=(c : Collection(T)) : Boolean c and $self$ are equal when both are maps of the same key and range types, and $c \rightarrow asSet() = self \rightarrow asSet()$.

<>(c : Collection(T)) : Boolean The negation of $=$.

size() : Integer

```
post: result = self->asSet()->size()
```

includesValue(object : T) : Boolean True if the $object$ is an element of the map range, false otherwise:

```
post: result = self->values()->includes(object)
```

includesKey(object : T) : Boolean True if the $object$ is an element of the map key set, false otherwise:

```
post: result = self->keys()->includes(object)
```

excludesValue(object : T) : Boolean True if the $object$ is not an element of the map range, false otherwise:

```
post: result = self->values()->excludes(object)
```

excludesKey(object : T) : Boolean True if the $object$ is not an element of the map domain, false otherwise:

```
post: result = self->keys()->excludes(object)
```

count(object : T) : Integer The number of times the $object$ occurs as an element of the map range (a bag):

```
post: result = self->values()->count(object)
```

includesAll(c2 : Collection(T)) : Boolean True if $c2$ is a map, and the set of pairs of $self$ contains all those of $c2$, false otherwise:

```
post:
  result = self->asSet()->includesAll(c2->asSet())
```

excludesAll(c2 : Collection(T)) : Boolean True if $c2$ is a map, and the set of pairs of $self$ is disjoint from those of $c2$, false otherwise:

```
post:
  result = self->asSet()->excludesAll(c2->asSet())
```

$isEmpty() : Boolean$, $notEmpty() : Boolean$ Defined as for general collections.

$max() : T$, $min() : T$, $sum() : T$ Defined as the corresponding operations on $self \rightarrow values()$.

$asSet() : Set(Tuple(first : K, second : T))$ The underlying set of pairs of the map. Since duplicate keys are not permitted, this has the same size as $self \rightarrow keys()$.

$keys() : Set(K)$ The set of keys in the map, ie., its domain:

```
post:
  result = self->asSet()->collect(p|p.first)->asSet()
```

$values() : Bag(T)$ The bag of values in the map, ie., its range:

```
post:
  result = self->asSet()->collect(p|p.second)
```

$restrict(ks : Set(K)) : Map(K, T)$ Domain restriction $ks \lhd self$. The map restricted to the keys in ks. Its elements are the pairs of $self$ whose key is in ks:

```
post:
  result->asSet() =
    self->asSet()->select(ks->includes(first))
```

$-(m : Map(K, T)) : Map(K, T)$ Map subtraction: the elements of $self$ that are not in m.

```
post:
  result->asSet() =
    self->asSet() - m->asSet()
```

$union(m : Map(K, T)) : Map(K, T)$ Map override, $self \oplus m$. The pairs of $self$ which do not conflict with pairs of m, together with all pairs of m:

```
post:
  result->asSet() =
    m->asSet()->union(
      self->asSet()->select(p |
         m->keys()->excludes(p.first)))
```

$intersection(m : Map(K, T)) : Map(K, T)$ The pairs of $self$ which are also in m:

```
post:
  result->asSet() =
    m->asSet()->intersection(self->asSet())
```

$including(k : K, v : T) : Map(K, T)$ The pairs of $self$, with the additional or overriding mapping of k to v:

$$self \rightarrow including(k, v) \; = \\ \quad self \rightarrow union(Map\{k \mapsto v\})$$

$excluding(k : K, v : T) : Map(K, T)$ The pairs of $self$, with any mapping of k to v removed:

$$self \rightarrow excluding(k, v) \; = \\ \quad self - Map\{k \mapsto v\}$$

$at(k : K) : T$ The value to which $self$ maps k, $null$ if k is not in $self \rightarrow keys()$:

```
post:
  (self->keys()->excludes(k) implies result = null) and
  (self->keys()->includes(k) implies
    result = self->restrict(Set{k})->values()->any())
```

any Defined as

$$m \rightarrow any(x \mid P) = m \rightarrow values() \rightarrow any(x \mid P)$$

Likewise for $forAll$, $exists$, one.

$select$ The map formed from the range elements which satisfy the $select$ condition:

$$m \rightarrow select(x \mid P(x)) \; = \\ \quad m \rightarrow restrict(m \rightarrow keys() \rightarrow select(k \mid P(m \rightarrow at(k))))$$

$reject$ The map formed from the range elements which do not satisfy the $reject$ condition:

$$m \rightarrow reject(x \mid P(x)) \; = \\ \quad m \rightarrow restrict(m \rightarrow keys() \rightarrow reject(k \mid P(m \rightarrow at(k))))$$

$collect$ Map composition (chaining). The map formed by composing the map with the evaluation of the $collect$ condition:

$$m \rightarrow collect(x \mid e(x)) \rightarrow asSet() \; = \\ \quad m \rightarrow keys() \rightarrow collect(k \mid \\ \quad\quad k \mapsto e(m \rightarrow at(k))) \rightarrow asSet()$$

$isUnique$ The map range composed with the expression produces a set, ie., the composed map is injective:

$$m \rightarrow isUnique(e) \; = \\ \quad m \rightarrow values() \rightarrow isUnique(e)$$

1 Implementation

Implementations of map operators for Java, C#, C++, Python and C may be found in the OCL libraries at http://www.nms.kcl.ac.uk/kevin.lano/libraries. Eg., ocl.py for Python.

2 Further operators

It would be useful to have map formation operators such as

$$s \rightarrow collect(x \mid e(x) \mapsto v(x))$$

to form a map from another collection s, and

$$m \rightarrow inverse()$$

to produce the inverse of an injective map m.