Assignment 2A Solution

Problem:

Consider the following specification and implementation of a method subtract belonging to the KnowledgeDistribution class.

```
/**
  * @require k != null and
  *          for each knowledge-state ks,
  *          this.weight(ks) >= k.weight(ks)
  * @ensure \old(k) = k and
  *          for each knowledge state ks,
  *               this.weight(ks) =
  *                    \old(this).weight(ks) - k.weight(ks).
  *
  */
  public void subtract(KnowledgeDistribution k) {
       Iterator<BigFraction> it = k.iterator();
       BigFraction ks = null;
       BigFraction w = null;
       while (it.hasNext()) {
            ks = it.next();
            w = this.weight(ks);
            this.subtract(ks, w);
            this.add(ks, w.subtract(k.weight(ks)));
       }
  }
```

State a loop invariant of the implementation (above) of the method, and use it to argue that the implementation is partially correct. Provide a further argument that the method is totally correct.

You may assume that the KnowledgeDistribution methods used in the implementation are correctly programmed according to their specifications as in Assignment 1.

Solution:

Loop invariant [3]:

The following is an invariant of the loop in question:

Using semi-formal notation:

```
\old(k) = k, and
if ks = null  then
     \old(this) = this,  and
if ks != null then
     for each knowledge state ks'
          if ks' <= ks then
               this.weight(ks') =
                    \old(this).weight(ks') - k.weight(ks')
          else if ks' > ks then
               this.weight(ks') = \old(this).weight(ks')
```

In words:

- The parameter k has not been modified and,
- if ks is null then the KnowledgeDistribution "this" has not been modified, and
- if ks is not null then for each knowledge-state ks' such that ks'<=ks, the weight of "this" at ks' is the same as its original value minus the weight of k at ks'. The weight of all other knowledge-states in "this" is unchanged.

The basic idea: The necessary subtractions have been made for all knowledge states up until the current position of the iterator. No other modifications have been made.

NOTE: assuming that the precondition of the method is satisfied when the method is called, it can be checked that this invariant holds

(i) prior to entering the loop for the first time, and
(ii) after each execution of the method body.

From this we know that if the loop terminates, it will terminate in a state satisfying the invariant.

Partial correctness [1]:

We need to show that if the method was invoked from a state satisfying its precondition (the @require clause) and it terminates, then it does so in a state satisfying its post-condition (the @ensure clause).

The method completes when the loop terminates, and so it is enough to show that if the loop terminates and the loop invariant holds, then the post-condition is met.

The basic idea: If the necessary subtractions have been made for all knowledge states up until the current position of the iterator, and the iterator has no more elements to iterator over, then all the necessary subtractions have been made, and so the post-condition has been established.

In more detail:

The loop terminates if it.hasNext() is false meaning that either:

1. ks is null and the support of k is empty OR
2. ks is not null and all elements in the support of k are less than or equal to ks

We verify each of these cases in turn.Case 1: If the invariant holds and (1) holds then the post-condition is met:

```
     invariant and
     ks =null and
     for all knowledge states ks,
          k.weight(ks) = 0
==>
     \old(k) = k and
     \old(this) = this and
     for all knowledge states ks,
          k.weight(ks) = 0
<==>
     \old(k) = k and
     for each knowledge state ks,
          this.weight(ks) = \old(this).weight(ks) and
          k.weight(ks) = 0
==>
     \old(k) = k and
     for each knowledge state ks,
          this.weight(ks) =
                    \old(this).weight(ks) - k.weight(ks)
<==>
     post-condition
```

Case 2: If the invariant holds and (2) holds then the post-condition is met:

```
     invariant and
     ks !=null and
     for each knowledge state ks'
          if ks' > ks then k.weight(ks') = 0
==>
     \old(k) = k, and
     for each knowledge state ks'
          if ks' <= ks then
               this.weight(ks') =
                    \old(this).weight(ks') - k.weight(ks')
          else if ks' > ks then
               this.weight(ks') = \old(this).weight(ks')
               and k.weight(ks') = 0
==>
     \old(k) = k, and
     for each knowledge state ks'
               this.weight(ks') =
                    \old(this).weight(ks') - k.weight(ks')

<=>
     post-condition
```

Total correctness: [1]

Since the method is partially with respect to its specification, it suffices to show that the method terminates from all initial states in which its precondition is met.

The basic idea:

The iterator `it` over the elements in the support of the unchanged parameter `k` is initialised prior to commencement of the loop and advanced using the `next` function once per iteration. Since the support of `k` is finite, this guarantees that the loop condition, `it.next()`, will eventually become false, and the loop will terminate [assuming that the loop body itself terminates, which it does].

Since the first three lines of the method are trivially terminating, this gives us that the method as a whole is terminating.