CS 784 Programming Languages
Fall 2011
Home Work 2 map ADT Solution

1. (40+20 points) The ADT known as Map (Cf. java.util.Map) has the following operations described only informally here:
   i. map: Yields the empty map.
   ii. clear: Takes a map m and yields the empty map.
   iii. containsKey: Takes a map m and a key k as input, and checks if m has k.
   iv. containsValue: Takes a map m and a value v as input, and checks if m has v bound to some key.
   v. equals: Takes two maps m and n as input, and checks if m and n are equivalent, that is, contain similar bindings.
   vi. get: Takes a map m and a key k as input, and yields the value bound to k in m.
   vii. isEmpty: Takes a map m as input, and checks if m is empty.
   viii. put: Takes a map m, a key k, and a value v as input, and yields a new map that behaves like m except that k is bound to v.
   ix. remove: Takes a map m and a key k as input, and yields a new map which has k deleted from m (if it were present).
   x. size: Takes a map m as input, and yields the number of key-value pairs in m.
   1. (40 points) Specify the syntax and semantics of the ADT Map using algebraic approach. Include a proof-like argument supporting the correctness of your specs.
   2. (20 points) Give an implementation of the ADT Map in Scheme. Include at least 5 thorough test cases.

ADT specs are written in functional style and as such do not support assignments. For example, put(m,k,v) does not change the state of the existing map m. Instead, conceptually, it creates a new map by "cloning" m and then modifying the copy by inserting the new k-v binding. Note further that: (a) The informal spec does not rule out (k, v1) and (k, v2) to be present in the map, unless we take the "the" in get-definition seriously, and we do. (b) Is get(m, k) always defined? (c) Some times put(m, k, v) = m, and hence size(put(m,k,v)) = 1 + size(m) is not always true.

Axioms:

i. clear(m) ≡ map()
ii. isEmpty(map()) ≡ true
iii. isEmpty(put(m,k, v)) ≡ false
iv. containsKey(map(), k) ≡ false
v. containsKey(put(m,j,v), k) ≡ (j == k or containsKey(m, k))
vi. containsValue(map(), v) ≡ false
vii. containsValue(put(m,j,v), k) ≡ (j == k or containsValue(m, k))
viii. get(map(), k) ≡ error
ix. get(put(m, j, v), k) ≡ if j == k then v else get(m, k) fi
x. remove(map(), k) ≡ map()
xi. remove(put(m, j, u), k) ≡ if j == k then remove(m, k) else put(remove(m, k), j, u) fi
xii. size(map()) = 0
xiii. size(put(m, k, v)) ≡ 1 + size(remove(m, k))
xiv. equals(m, m) ≡ true
xv. equals(m, n) ≡ equals(n, m)
xvi. equals(put(m, j, u), put(n, k, v)) ≡

```
if (j == k and u == v)
  then equals(remove(m, j), remove(n, k))
  else equals(put(remove(m, k), k, v), put(remove(n, j), j, u))
fi
```

Proof-like argument: See also http://www.cs.wright.edu/~pmateti/Courses/784/Exams/784-2010-Fall-midterm-soln.html

Clearly, any arbitrary map can be constructed starting from map() and then put-ting the needed key-value pairs.

Axioms (Ai) (Aii) follow from intuitive defs (Di) and (Dii). Aii, Aiv, Avi, Aviii, Ax, Axii capture emptiness. Aiii captures non-emptiness. Av, Avii, Axiii capture the notion of adding an item provided it is not already there.

Equals as we have it here ought to have the reflexive, symmetric and transitive properties of mathematical equality. Axiv, Axv provide the first two. Transitivity of our equals is left as an exercise to you.

What is the difference between clear(m) == map() versus equals(map(), clear(m))? The former permits substitution of rhs for lhs; the latter does not (formally speaking).

Completeness: We make sure all functions/predicates are well-defined for all constructor expressions. We also make sure that all map-expressions are reducible to pure constructor forms by one or more substitutions for lhs by the rhs from the above equalities.

Soundness: Since we already know what maps are -- i.e., independently of the algebraic specs -- we can "see" that there are no unsound axioms. This does require formal proof, but it is beyond the scope of our CS784 as it requires "interpretations and models" (from mathematical logic) of maps.

Minimality of the given set of axioms is also generally expected to be proven. That is, if we drop even one of the given axioms there will be incompleteness. In this example, it is just a tedious exercise. We can show that it is minimal by dropping one of the axioms at a time, and show that the dropped axiom cannot be deduced from the others.

Note that a different set of axioms might be smaller in size.

Implementation: **(a)** There is already a hash table data structure available in Scheme. You can we use it and provide wrappers over it (to suit the operations asked in this Homework. Since the implementation is left to the implementers, the underlying choice shouldn't matter as long as it sticks to the semantics we described in the ADT. (b) Design an alternate data structure. (c) What is interesting is that we can take the above axioms themselves as "rewrite rules" and produce an implementation. All these are left as exercises to you.