Constructing the Permutations of a Set

A permutation of a set $S$ is a sequence with no repetitions containing every member of $S$ (and no other values). For instance, the sequence $\langle 3, 2, 5, 1, 0, 4 \rangle$ is a permutation of the set $\{0, 1, 2, 3, 4, 5\}$.

As we have seen, the factorial function counts the permutations of a set of a specified cardinality. The factorial is defined by the recursion equations

$$0! = 1,$$
$$\frac{(n + 1)!}{n!} = (n + 1).$$

These equations hint at a possible algorithm for constructing the permutations of a set $S$: If $S$ is empty, its only permutation is the empty sequence. Otherwise, choose the first element $s$ of a permutation in every possible way (taking $s$ to be each member of $S$ in turn); then construct every permutation of $S - \{s\}$ and extend every such permutation by adding $s$ at the front.

1. The definition of the process-permutations procedure in the “Permutations and Combinations” handout (which depends on the (discrete sets) library) shows one way of using this algorithm to apply a procedure proc to every permutation of the elements of a set aro.

Using this procedure as a model, define and test a Scheme procedure set-permutations that takes any set aro as its only argument and returns a set of all of the permutations of aro. (Hint: Use map or set-map instead of for-each and consider how to combine the results of the recursive calls.)

Constructing the Combinations of a Set

A combination of $k$ members from a set $S$ of cardinality $n$ is simply a $k$-member subset of $S$. We have seen that the binomial coefficient $\binom{n}{k}$ counts such combinations.

2. Define and test a Scheme procedure set-combinations that takes a set $S$ and a natural number $k$ as arguments and returns the set of all the $k$-member combinations of members of $C$. The reading “Permutations and Combinations” contains a prose description of the algorithm to use.

The (discrete bags) Library

The (discrete bags) library provides a collection of support procedures for working with bags. Take a moment to look it over before proceeding. Many of the operations are quite similar to their counterparts in the (discrete sets) library, but there are a few new ones — multiplicity, set->bag, and bag->set.

Bag Permutations

Now let’s consider combinatorial constructions starting with a bag rather than a set. Since sequences, like bags, can contain duplicates, it makes sense to construct permutations from the members of bags. For instance, $\langle b, a, b, b, c \rangle$ is a permutation of the bag $\{a, b, b, c\}$.

However, when a bag contains duplicate members, the number of possible permutations is less than the number of permutations of a set of the same cardinality, because rearranging duplicates does not change the identity of a permutation — they are still the same values in the same order, so it’s still the same sequence. In the extreme case, a bag consisting of nothing but duplicates, such as $\{k, k, k, k\}$, has only one permutation (namely $\langle k, k, k, k \rangle$), not 4!. 
If we adapt the algorithm for constructing set permutations that we implemented above in a naive way, our implementation could spend a lot of time constructing duplicate permutations, only to discard all of those duplicates when we assemble the permutations into a set. It would be better not to construct the duplicate permutations to begin with. Fortunately, in adapting the set-permutations algorithm to bags, we can add a step to ensure that duplicate permutations are never constructed in the first place, no matter how many duplicate values there are in the bag.

3. Figure out what that change is and implement it in order to define a bag-permutations procedure.

**Counting Bag Permutations**

The Bookkeeper Rule in the textbook provides a way to count bag permutations: The factorial of the sum of the multiplicities of the values in the bag, divided by the product of the factorials of those multiplicities.

4. Write and test a definition of a bag-permutation-count procedure that takes any bag as its argument and returns the number of permutations of the members of that bag.

5. Use your rule to count the permutations of a bag containing 12 as, 8 bs, 3 cs, and 1 d.