Exercise 2.82.

Show how to generalize `apply-generic` to handle coercion in the general case of multiple arguments. One strategy is to attempt to coerce all the arguments to the type of the first argument, then to the type of the second argument, and so on. Give an example of a situation where this strategy (and likewise the two-argument version given above) is not sufficiently general. (Hint: Consider the case where there are some suitable mixed-type operations present in the table that will not be tried.)

Answer.

The following `apply-generic` procedure handles coercion in the general case of multiple arguments.

```
(define (apply-generic op . args)
  (define (coerce-all type items)  ;; coerce all the items in a list to a type
    (if (null? items)
        '()
        (let ((type1 (car items))
              (a1 (car items)))
          (let ((t1->type (get-coercion type1 type)))
            (if (t1->type a1)
                (cons (t1->type a1)
                      (coerce-all type (cdr items)))
                (coerce-all type (cdr items)))))))
  (define (coerced-succeeded? items)  ;; determine whether all the items in the
    (= (length items)                 ;; original list 'args' have been coerced to
       (length args)))                ;; the same type
  (let ((type-tags (map type-tag args)))
    (let ((proc (get op type-tags)))
      (if proc
          (apply proc (map contents args))
          (let ((coerced-lists (filter coerced-succeeded?
                                       (map (lambda (arg)
                                              (coerce-all (type-tag arg) args))
                                            args))))
            (if (not (null? coerced-lists))
                (for-each (lambda (l)
                            (apply-generic op . l))
                          coerced-lists)
                (error "No method for these types"
                       (list op type-tags)))))))))
```

The generalized `apply-generic` procedure performs as originally if all the elements in a list have the same type. Otherwise, it maps the list into a bounch of lists in which all the elements were coerced to the type of the first element, then to the type of the second argument, and so on, as shown in Figure 1. It then selects those lists in which all the elements have been successfully coerced to a single type. Finishing all the steps above, we accomplishs the task by applying `apply-generic` procedure to each of the filtered lists with the designated operation.

There are several cases where this strategy might fail. For example, suppose there is a list which contains a real number as the first element and all the subsequent elements of complex numbers. We are now required to stretch the magnitude of all the complex numbers by the value of the real number. Performing this operation using the `apply-generic` procedure above will surely go to a wrong answer. Because the `apply-generic` procedure here only indexes operations of a single type, rather than the ones of mixed-type which the problem belongs to.

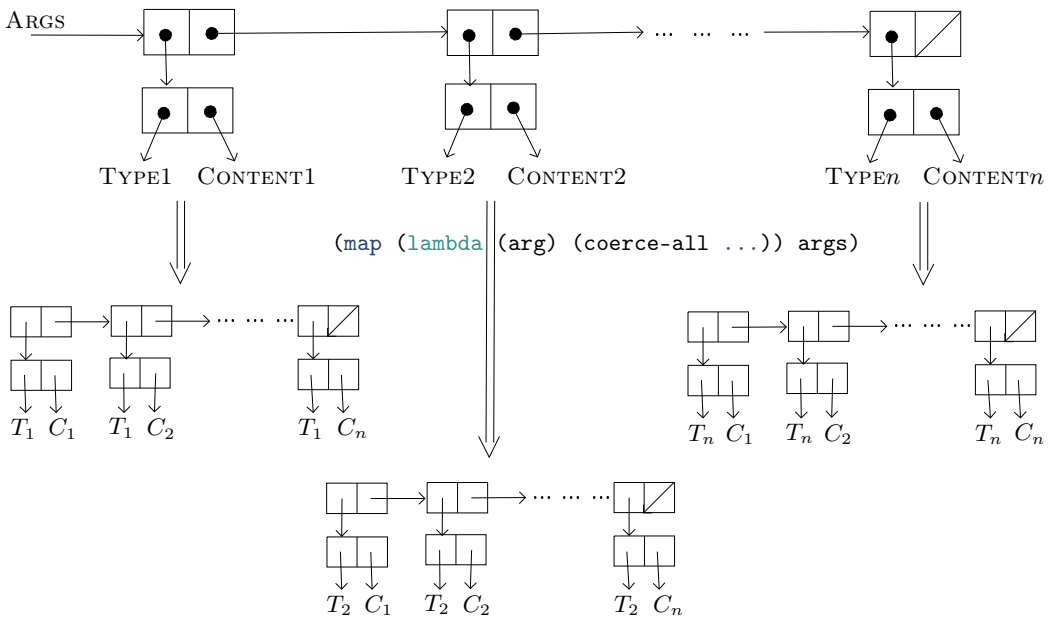
ARGS
TYPE1 CONTENT1
TYPE2 CONTENT2
TYPEn CONTENTn
(map (lambda (arg) (coerce-all ...)) args)
T1 C1 T1 C2 T1 Cn
Tn C1 Tn C2 Tn Cn
T2 C1 T2 C2 T2 Cn

Figure 1.