Exercise 2.88.

Extend the polynomial system to include subtraction of polynomials. (Hint: You may find it helpful to define a generic negation operation.)

Answer.

The plain way to performing subtraction between two polynomials is almost identical to that one we followed in implementing addition operation, except that the resulting terms list is produced by subtraction. Besides, we can also subtract two polys by adding the negation of the subtracter onto the minuend one.

We begin by implementing the plain strategy, just follow the way of addition:

```
(define (install-polynomial-package)
  ;; internal procedures
  <representation of poly, terms and term lists>
  ...
  <other internal procedures>
  ...
  (define (sub-poly p1 p2)
    (if (same-variable? (variable p1) (variable p2))
        (make-poly (variable p1)
                   (sub-term (term-list p1)
                             (term-list p2)))
        (error "Polys not in same var -- SUB-POLY"
               (list p1 p2))))
  (define (sub-terms L1 L2)
    (cond ((empty-termlist? L1) L2)
          ((empty-termlist? L2) L1)
          (else
            (let ((t1 (first-term L1))
                  (t2 (first-term L2)))
              (cond ((> (order t1) (order t2))
                     (adjoin-term t1
                                  (sub-terms (rest-terms L1) L2)))
                    ((< (order t1) (order t2))
                     (adjoin-term t2
                                  (sub-terms L1 (rest-terms L2))))
                    (else
                      (adjoin-term (make-term (order t1)
                                              (sub (coeff t1) (coeff t2)))
                                   (sub-terms (rest-terms L1)
                                              (rest-terms L2)))))))))

  ;; interface to rest of the system
  ...
  <other interface procedures>
  ...
  (put 'sub '(polynomial polynomial)
       (lambda (p1 p2) (tag (sub-poly p1 p2))))
  'done)
```

Now, let's come to erect the `sub-poly` procedure by means of `add-poly` together with a generic negation operation:

```
(define (install-polynomial-package)
  ;; internal procedures
  <representation of poly, terms and term lists>
```

```
...
<other internal procedures>
...
(define (sub-poly p1 p2)
  (add-poly p1 (neg p2))
(define (neg p)
  (make-poly (variable p)
             (neg-terms (term-list p))))
(define (neg-terms L)
  (if (empty-termlist? L)
      (the-empty-termlist)
      (let ((t1 (first-term L)))
        (adjoin-term (make-term (order t1)
                                (neg (coeff t1)))
                     (neg-term (rest-terms L))))))

;; interface to rest of the system
...
<other interface procedures>
...
(put 'sub '(polynomial polynomial)
     (lambda (p1 p2) (tag (sub-poly p1 p2))))
(put 'neg 'polynomial
     (lambda (p) (tag (neg p))))
'done)
```

We also have to install various negation procedures into Scheme-number, rational, real and complex packages:

```
;; inside the Scheme-number package
(define (neg x) (- x))
(put 'neg 'scheme
     (lambda (x) (tag (neg x))))

;; inside the rational package
(define (neg x)
  (make-rat (- (numer x))
            (denom x)))
(put 'neg 'rational
     (lambda (x) (tag (neg x))))

;; inside the real package
(define (neg x)
  (make-real (- x)))
(put 'neg 'real
     (lambda (x) (tag (neg x))))

;; inside the complex package
(define (neg z)
  (make-from-real-imag (- (real-part z))
                       (- (imag-part z))))
(put 'neg 'complex
     (lambda (z) (tag (neg z))))
```

Users can access the `neg` procedure by means of the following procedure:

```
(define (neg x) (apply-generic 'neg 'x))
```