Generic Arithmetic Package[1]

Summary and Exercise Answer

- ### Ordinary Numbers
    - Summary: Too simple
    - Exercise:
        - 5.1A - simple wrap operation

- ### Rational Numbers
    - Summary:
        - **constructor**: make-rat
        - **selector**: numer & denom
        - **basic procedures**: do the operations separately on the numer part and denom part

    - Exercises:
        - **5.2**: [[961] [169]]
        - **5.3A**: equ-rat? -- type is (RepRat X RepRat) -> Bool

- ### Operations across different types
    - Summary:
        - **Simple coerce:**
          Change number n to rational number n/1

    - Exercise:
        - **5.4A**:
          Core procedure is --

          ```
              (define (repnum->reprat num)
                (make-rat (create-number num) (create-number 1)))
          ```

          Explanations:

          1. This procedure is in the ps5-code.scm file, included in the

```
(define (RRmethod->NRmethod method)
  (lambda (num rat)
    (method
     (repnum->reprat num)
     rat)))
```

procedure, and the *RRmethod->NRmethod* procedure will be called in when you use *apply-generic*, and this procedure will tear down the tag of the number.
In the end, the *num* in *repnum->reprat* procudure should be pure number, without tag. And this is the most tricky part in this exercise.

2. So you need to know the type of the parameter when you are defining the procedure, especially the one will be used in the *apply-generic*.

- **5.4B**: In the code
  Tips:
  create-rational's parameter should be added tag:

```
(define r5/13 (create-rational (create-number 5) (create-number 13)))
```

- **Polynomials**
  - **Summary**:
    - The package has defined the *add*, *mul*, and *=zero?* procedures.
    - The terms are arranged in high order
    - The inner data structure is **term list**, cooperated with operations like: *the-empty-termlist* and *adjoin-term* and selectors.
    - The polynomial here is a cons of one variable and one termlist.
  - **Research on the Code** and **the Exercises' Answer**:
    - > Keep clear of the data types:

      1. The core data type is **RepPoly** and it's tag is **polynomial**
      2. **RepPoly** = **Variable X RepTerms**
      3. RepTerms could be created by List(GN), for the coefficients here are generic numbers

- > dense/coeffs->sparse/terms

  This procedure has a inner iteration called **dt->st**, which reverse the coefficients list parameter iterate through it. It checks the value of each coefficient, the one with 0 coefficient will be passed, and the one with non-zero value, the dt->st procedure will create a new term and add it behind the term-list it keep. Once it met the end of the coeff-list's end, it'll return the term list.

- > *terms L1 L2

  This procedure is very complex. First if you do the multiplication among two polynomials, you will use two basic procedure:

  1. *-term-by-all-terms
  2. +terms

  And the first procedure is very important, give it the first element in L1 and it will multiply it by all the elements in L2

- **5.5A**:

  > map-term is a procedure that applies it's procedure parameter to every term in the termlist parameter.

  Check this procedure in the ps5-code.scm

- **5.5B**:

  > use map-coeffs to wrap the input coeffs list

- **5.5C**:

  > dont understand the meaning of "pretty-printed"

- **5.6A**:

  > This question focus on the type coercing problem.

- **5.6B**:

  > Simple

- ## Completing Polynomial Package

  - Add negate and div operation
  - **Exercise 5.7A:**
    In order to write the negate-polynomial, we need to build the procedure like this:
    map-terms->negate-terms->negate-polynomial
  - **Exercise 5.7B**
    equ-polynomial
    Just like the hardware implementation, we use the feature:

    ```
    if (p1 == p2) then (p1-p2 == 0)
    ```

- ## More Operations across Different Types

  - In this part, we need to add more datatype coerce operations
  - **Exercise 5.8A**: In order to implement the repnum->reppoly, first we need to know how could a generic number become a polynomial:

    > 3(generic number) --> 3 X x^0 (polynomial)
    > Then, the building function is simple.

  - **Exercise 5.8C**:
    This question is about multiply a polynomial by a rational number.

    1. Should we just turn polynomial p into p/1?
       > If we use p/1, then the multiplication procedure will be rational X rational, there will be far more operations.
       > For example:
       >
       > $$\frac{p}{1} = \frac{a+b+c}{1}$$
       >
       > and
       >
       > $$\frac{p}{1} \times \frac{a}{b} = \frac{b \times p + a \times 1}{b}$$
       >
       > So the total operations will be:
       >
       > $$3 \times 1(3\ mul\ operations) + 4(1\ add\ operation) + 3 \times 1(3\ div\ operations) = 10$$
    2. What about coerce the rational number to polynomial?

> In the last case, the total operation will be:
>
> $$3\times 1+2\times 1=5$$

- **Polynomial Evaluation**
  - This part wants us to use the polynomial like a polynomial function:
    It's quite simple because we could just use the add operation to finish this task

---

1. Author: 赵睿哲 id: 1200012778 email: vincentzhaorz@pku.edu.cn↩