Structure and Interpretation of Computer Programs: Assignment 2

Seung-Hoon Na

November 17, 2020

1 Table ADT

(아래 2개의 문제에 대한 구현이 모두 포함된 hash-table.rkt파일을 제출하시오. 실행후에 Problem 1.2의 test 결과가 나와야 함)

Hash table을 이용하여 Table ADT를 구현하시오. 단, Lect12.pdf slide의 내용을 참조하고, N개의 buckets은 racket의 make-vector를 이용하여 생성하시오. (주의: Racket에서 제공되는 hash-table을 이용하면 안되고, 모두 직접 구현 필요)

```
(define table-tag 'hash-table)
(define (make-table size hashfunc)
    (let ((buckets (make-buckets size nil)))
    (list table-tag size hashfunc buckets)))

(define (size-of tbl) (cadr tbl))
(define (hashfunc-of tbl) (caddr tbl))
(define (buckets-of tbl) (cadddr tbl))
```

1.1 Problem 1:make-table, table-get과 table-put!

Table ADT를 위한 생성자 make-table, Accessor를 위한 프로시저들인 table-get과 table-put!를 구현하시오.

```
(define (make-table size hashfunc)
...
(define (table-get tbl key)
...
(define (table-put! tbl key val)
...
```

1.2 Problem 2: Test code 작성

구현된 Table ADT 연산을 test하기 위해 symbol 또는 symbol의 리스트를 key로 하는 pair들을 table-put!를 통해 추가하고, 추가된 key들을 table-get을 통해 얻어오는 test code를 작성하고, 해당 code의 실험 결과를 기술하시오.

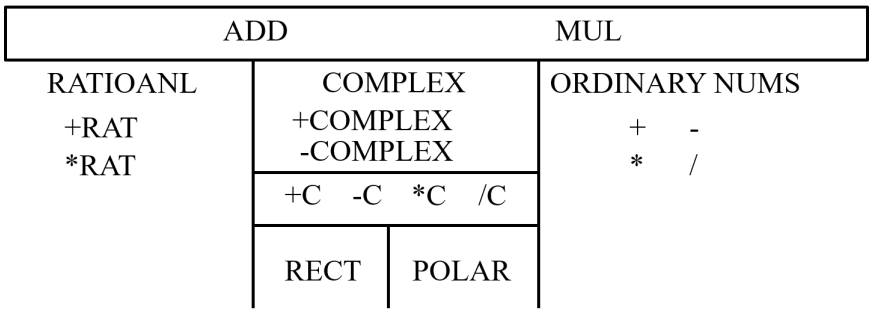
2 Generic Arithmetic System

(아래 4개의 문제에 대한 구현이 모두 포함된 arithemtic.rkt파일을 제출하시오. 실행후에 Problem 2.4의 test 결과가 나와야 함. hash-table.rkt의 내용은 arithemtic.rkt에 포함시켜도 무방함.)

다음 그림과 같이 실수 (ordinary number), 유리수 (Rational number)와 복소수 (Complex number) 세 가지 유형의 number에 대한 일반화된 연산자인 ADD, MUL를 구현하시오. Lect10.pdf slide 및 교과서 2.5절 내용을 참조하여, 각 유형의 number를 위해 tagged data를 사용하고, 연산의 결과도 tagged data가 리턴되도록 하라.

예를 들어, ADD연산에서 arguments의 유형과 결과값의 유형은 다음과 같다.

1. 유리수 + 유리수 → 유리수
2. 복소수 + 복소수 → 복소수
3. 유리수 + 실수 → 실수
4. 복소수 + 실수 → 복소수



2.1 Problem 1: Tagged data구현

각 유형별 필요한 tagged data의 생성자 및 연산자들을 모두 구현하시오.

예를 들어, 유리수의 경우에 필요한 프로시저들은 다음과 같다.

```
(define (make-rat x y) (attach-type 'rational (cons x y)))

(define (+rat x y)
(make-rat (+ (* (numer x) (denom y))
(* (denom x) (numer y)))
(* (denom x) (denom y))))

(define (*rat x y) ...)
```

복소수의 경우에는 두 개의 tag를 사용하여 rectangular 및 polar 좌표계에 따라 각각에 대한 tagged data를 정의하고 이에 해당되는 생성자 및 연산자들도 별도로 구현해야 한다.

복소수의 경우, rectangular 및 polar 좌표계 각각을 위해 필요한 기본 프로시저들의 예는 다음과 같다.

```
(define (make-rectangular x y)
(attach-type 'rectangular (cons x y)))
(define (real-part-rectangular z) (car z))
```

```
(define (imag-part-rectangular z) (cdr z))
(define (magnitude-rectangular z)
    (sqrt (+ (square (car z))
             (square (cdr z)))))
(define (angle-rectangular z)
        (atan (cdr z) (car z)))


(define (make-polar r a)
(attach-type 'polar (cons r a))
(define (real-part-polar z)
(* (car z) (cos (cdr z))))
(define (imag-part-polar z)
(* (car z) (sin (cdr z))))
(define (magnitude-polar z) (car z))
(define (angle-polar z) (cdr z))
```

추가로 두 좌표계를 포괄하는 최종적인 복소수의 생성자와 연산자들은 다음과 같다.

```
(define (make-complex z) (attach-type 'complex z))
(define (make-complex-rectangular x y) ...
(define (make-complex-polar r a) ...

(define (+complex z1 z2) (make-complex (+c z1 z2)))
(define (-complex z1 z2) (make-complex (-c z1 z2)))
```

ordinary 실수의 경우에는 tag로 ′scheme − number를 부여한 tagged data를 구현하면 되고, 필요한 생성자의 예는 다음과 같다 (연산자의 경우에는 primitive procedures를 직접 사용).

```
(define (make-scheme-number n) ...
```

2.2 Problem 2: Package installation procedure 구현

Problem 1에서 구현된 Table ADT를 사용하여 package installation을 위한 global table을 생성하고 세 가지 유형별로 필요한 프로시저들을 적절한 key,value를 정의하여 등록하는 다음 각 package installation 프로시저를 구현하시오. (교과서 2.5절 내용 참고하여 problem 2.1의 필요한 프로시저들을 모두 각 유형별 internal procedure로 구현해도 무방함)

```
(define (install-scheme-number-package)
 ...
(define (install-rational-package)
 ...
(define (install-complex-package)
...
```

2.3 Problem 3: Generic arithmetic procedures 구현: apply-generic

교과서 2.5절 내용 참고하여 다음과 같이 ADD, MUL 를 정의하고 필요한 apply-generic 를 구현하시오. (교과서의 내용을 변경해서 구현해도 무방함)

```
(define (ADD x y) (apply-generic 'ADD x y))
(define (MUL x y) (apply-generic 'MUL x y))
```

2.4 Problem 4: Test code작성

구현된 Generic arithmetic procedures를 test하기 위해 몇 가지 실수, 유리수, 복소수를 생성하고 이들을 연산하여 그 결과를 출력하는 test code를 작성하고, test 수행한 결과를 보이시오.

다음은 number 생성자 수행 test 예시이다.

```
(define c1 (make-complex-rectangular 3 5))
(define c2 (make-complex-rectangular 1 2))

(define c3 (make-complex-polar 3 5))
(define c4 (make-complex-polar 1 2))

(define r1 (make-rat 3 5))
(define r2 (make-rat 5 6))

(define n1 (make-scheme-number 5))
(define n2 (make-scheme-number 8))
```

다음은 연산자 적용 test 예시이다.

```
(ADD c1 c2)
(ADD c1 c4)
(MUL c1 c4)
(MUL c3 c4)
(ADD r1 r2)
(MUL r1 r2)
(ADD n1 n2)
(MUL n1 n2)

....
(ADD c1 r1)
(MUL c3 r1)
(MUL c3 n2)
(MUL r2 n2)

...
```