MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Fall Semester, 1996

Lecture Notes, September 19 – Data Abstraction

Pair Abstraction

1. Constructor

   ; cons: T, T → Pair
   ```
   (cons <x> <y>) -> given x & y parts,
                     create a new Pair object
   ```

2. Accessors

   ; car, cdr: Pair → T
   ```
   (car <Pair>) -> the first part of the pair
   (cdr <Pair>) -> the second part of the pair
   ```

3. Contract

   ```
   (car (cons <x> <y>)) = <x>
   (cdr (cons <x> <y>)) = <y>
   ```

4. Abstraction Barrier
   Say nothing about representation or implementation of pairs!

Rational Number Abstraction

1. Constructor

   ; make-rat: Int, Int → RepRat
   ```
   (make-rat <n> <d>) -> <RepRat>
   ```

2. Accessors

   ; numer, denom: RepRat → Int
   ```
   (numer <RepRat>)
   (denom <RepRat>)
   ```

3. Contract

   ```
   (numer (make-rat <n> <d>)) = <n>
   (denom (make-rat <n> <d>)) = <d>
   ```

4. Abstraction Barrier
   Say nothing about representation or implementation of RepRat!

5. Representation & Implementation

```
(define (make-rat n d) (cons n d))
(define (numer r) (car r)
(define (denom r) (cdr r)
```

Using the Rational Number Abstraction

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

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

A “Rationalizing” Implementation

```
(define (numer r)
  (let ((g (gcd (car r) (cdr r))))
    (/ (car r) g)))

(define (denom r)
  (let ((g (gcd (car r) (cdr r))))
    (/ (cdr x) g)))

(define (make-rat n d)
  (cons n d))

(define (gcd a b)
  (if (= b 0)
    a
    (gcd b (remainder a b))))
```

Alternative “Rationalizing” Implementation

```
(define (numer r) (car r))

(define (denom r) (cdr r))

(define (make-rat n d)
  (let ((g (gcd n d)))
    (cons (/ n g)
          (/ d g))))
```