ФУНКЦИОНАЛНО ПРОГРАМИРАНЕ

Магдалина Тодорова
magda@fmi.uni-sofia.bg
todorova_magda@hotmail.com
кабинет 517, ФМИ

Тема 5

Абстракция с данни

1. Съставни структури от данни

Обект на следващите разглеждания са съставните структури от данни: рационални и комплексни числа, списъци, дървета, графи и др. Използването на съставни структури от данни в езика Scheme повишава концептуалното ниво, на което се описват програмите, увеличава модулността на означенията, повишава изразителната сила на езика.

Както възможността да се дефинират процедури позволява да се работи с процеси от по-високо концептуално ниво от това на примитивните оператори на езика, така възможността за дефиниране на съставни типове от данни позволява да се работи с данни от по-високо концептуално ниво от това на примитивните структури от данни на езика.

1. Съставни структури от данни

Всяка данна от съставен тип се състои от компоненти, които могат да са от различни типове. Намирането на компонентите на съставния тип се осъществява чрез специални процедури, наречени ***селектори***.

Ако са зададени отделните елементи, които изграждат стойност от някакъв съставен тип, тази съставна стойност може да бъде построена от елементите й с помощта на специални процедури, наречени ***конструктори***.

2. Подходът абстракция с данни

За създаване и работа със съставни структури от данни се прилага подходът ***абстракция с данни***.

Дефиниция: Абстракция с данни е методология, позволяваща да се отделят методите за използване на данните от методите за тяхното конкретно представяне.

Представяне на данните

конструктори, селектори

Използване на данните

методи за използване

2. Подходът абстракция с данни

Основна идея:

Програмите се конструират така, че да работят с „абстрактни данни“, чиято структура може да не е (напълно) уточнена.

След това представянето на данните се конкретизира с помощта на множество процедури, наречени конструктори и селектори, които реализират тези абстрактни данни по определения от автора конкретен начин.

2. Подходът абстракция с данни

Предимства:

✓Абстракцията с данни позволява съответните програми да се дефинират, използват и модифицират значително по-лесно. При промяна на представянето на данните се променят малки части от програмата – само тези процедури, които зависят от представянето на данните (конструкторите и селекторите).

✓Абстракцията с данни позволява да се работи в термините на съответната предметна област или разглеждания модел на съответната област. Работи се в термините на областта, а не в термините на съответното представяне, т.е. на подходящо ниво на абстракция.

2. Подходът абстракция с данни

Задача.

Да се напише програма, която да може да събира, изважда, умножава, дели, сравнява за равно, извежда рационални числа.

За решаване на задачата е необходимо да се решат следните две подзадачи:

1) Как да се представят рационалните числа? Какви са примитивните операции за тях?

2) Дефиниране на процедури за работа с рационални числа.

2. Подходът абстракция с данни

1) ***Как да се представят рационалните числа? Какви са примитивните операции за тях?***

Тъй като рационалното число е частно на две цели числа, представящи числителя и знаменателя му, възможни са следните представяния:

а) *като две цели числа, които не са свързани*

В този случай, за намиране например на сумата на две рационални числа трябва да се дефинират две процедури: едната за получаване на числителя, а другата – на знаменателя на сумата.

Това представяне е неудобно, защото е необходимо да се пази следа за това, кой числител на кой знаменател съответства.

2. Подходът абстракция с данни

б) *като двойка от цели числа (p, q), където p означава числителя, а q – знаменателя (q е различно от 0).*

При това представяне числителят и знаменателят са „залепени“ и образуват съставен обект, който наподобява ***точкова двойка***.

Избираме това представяне на рационални числа за решението на задачата.

В този случай примитивните операции са следните:

• от две дадени цели числа да се конструира рационално число;

• да се определи числителят на дадено рационално число;

• да се определи знаменателят на дадено рационално число.

2) Дефиниране на процедурите за работа с рационални числа

Нека имаме процедура за конструиране на рационално число по дадени числител ***n*** и знаменател ***d*** (*d* е различно от 0):

(make-rat n d) ⟶ конструира рационалното число **n/d**;

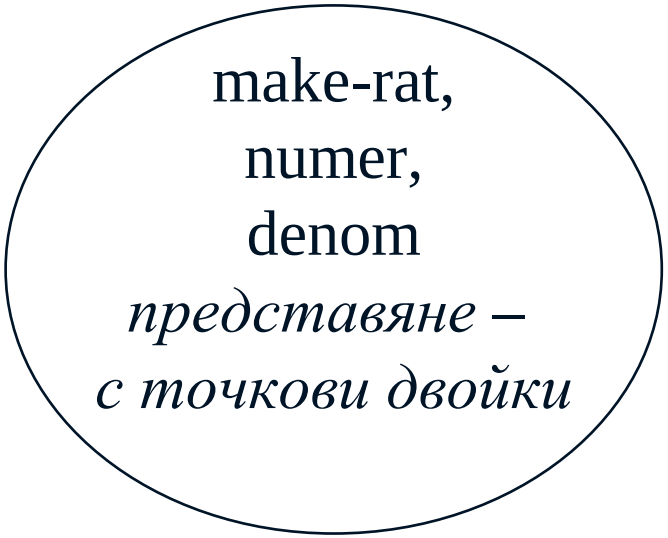
Нека имаме и процедури за извличане на числителя и знаменателя на дадено рационално число ***x***:

(numer x) ⟶ намира числителя на рационалното число **x**;
(denom x) ⟶ намира знаменателя на рационалното число **x**.

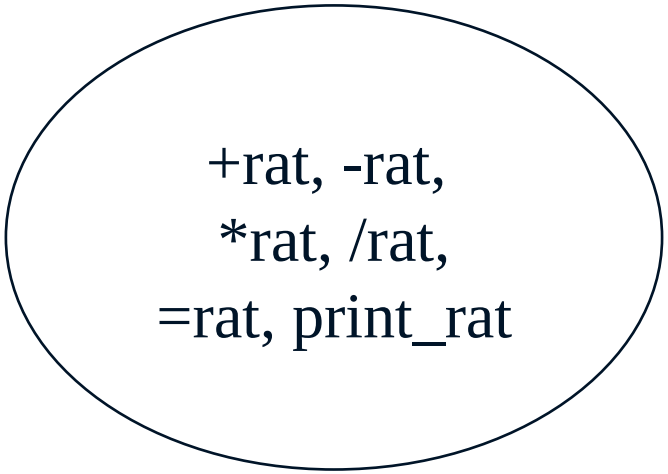
2. Подходът абстракция с данни

Подходът абстракция с данни за рационални числа

Представяне на данните



Използване на данните



2. Подходът абстракция с данни

За дефиниране на процедурите за работа с рационални числа ще приложим следните правила за рационално-числова аритметика:

$$\frac{n_1}{d_1} + \frac{n_2}{d_2} = \frac{n_1 d_2 + n_2 d_1}{d_1 d_2},$$

$$\frac{n_1}{d_1} - \frac{n_2}{d_2} = \frac{n_1 d_2 - n_2 d_1}{d_1 d_2},$$

$$\frac{n_1}{d_1} \cdot \frac{n_2}{d_2} = \frac{n_1 n_2}{d_1 d_2}, \quad \frac{n_1/d_1}{n_2/d_2} = \frac{n_1 d_2}{n_2 d_1},$$

$\frac{n_1}{d_1} = \frac{n_2}{d_2}$ тогава и само тогава, когато $n_1 d_2 = n_2 d_1$.

Дефиниции на процедурите за работа с рационални числа

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

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

Дефиниции на процедурите за работа с рационални числа

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

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

(define (=rat x y)
  (= (* (numer x) (denom y)) (* (denom x) (numer y))))
```

2. Подходът абстракция с данни

Дефинирахме процедурите +rat, -rat, *rat, /rat и =rat, които могат да се използват за работа с рационални числа.

За дефинирането им използвахме наготово процедурите: make-rat, numer и denom.

За да се използват процедурите +rat, -rat, *rat, /rat и =rat, трябва да се реализират процедурите: make-rat, numer и denom.

За това е необходимо да има средство, чрез което да се „слепят“ числителят и знаменателят, за да се образува рационално число.

Такова средство е ***точковата двойка*** – вградена в езика Scheme универсална структура от данни.

3. Точкова двойка

Синтаксис:

(obj1 . obj2)

За тази съставна структура са дефинирани:

- примитивен конструктор **cons**;
- примитивни селектори **car** и **cdr**.

3. Точкова двойка

Конструктор на точкова двойка е примитивната процедура ***cons***. Тя има два аргумента и връща съставен обект, който се състои от две части (два елемента), съвпадащи с оценките на аргументите на ***cons***.

(cons <obj1> <obj2>) → ([<obj1>] . [<obj2>])

Селектори на компонентите на точковите двойки са примитивните процедури ***car*** и ***cdr***.

3. Точкова двойка

Селектори на компонентите на точковите двойки са примитивните процедури ***car*** и ***cdr***.

Това са процедури с един аргумент, чиято оценка трябва да е точкова двойка.

car **–** извлича първия елемент на точкова двойка;

cdr **–** извлича втория елемент на точкова двойка.

(car <pair>) **(cdr <pair>)**

Примери:

```
> (define x (cons 1 2))
x
> x
(1 . 2)
> (car x)
1
> (cdr x)
2
> (define y (cons 3 4))
y
```

Примери:

```
> (define z (cons x y))
z
>z
((1 . 2) . (3 . 4))
> (car (car z))
1
> (cdr (car z))
2
>(car (cdr z))
3
>(cdr (cdr z))
4
```

4. Реализация на конструктора и селекторите

Представяне на рационалните числа в примерната система за работа с рационални числа

Процедурите make-rat, numer и denom могат да се реализират по следния начин:

```
(define (make-rat n d)                ; d е различно от 0
    (cons n d))
(define (numer x)
    (car x))
(define (denom x)
    (cdr x))
```

където d е различно от 0, цяло число.

Реализация на процедурата за извеждане

Процедурата за извеждане на рационално число има вида:

```
(define (print-rat x)
    (newline)
    (display (numer x))
    (display "/")
    (display (denom x)))
```

Пример:

```
>(print-rat (make-rat 1 5))

1/5
```

Експерименти

Пример.

```
>(define one-half (make-rat 1 2))
one-half
>(define one-third (make-rat 1 3))
one-third
> (print-rat (-rat one-half one-third))
1/6
> (print-rat (+rat one-third one-third))
6/9
```

Корекция на реализацията на конструктора

Както показва последният пример, направената реализация не съкращава рационални числа. Този недостатък може да се преодолее като се дефинира нов вариант на make-rat.

```
(define (make-rat n d)
    (if (= n 0) (cons 0 1)
        (let ((g (gcd (abs n) (abs d))))
            (if (or (and (>  n 0) (> d 0))
                    (and (< n 0) (< d 0)))
                (cons (/ (abs n) g) (/ (abs d) g))
                (cons (- (/ (abs n) g)) (/ (abs d) g))))))
```

където d е различно от 0, а gcd е НОД на две естествени числа, различни от 0.

Нива на абстракция

Приложения, използващи програмата за работа с рационални числа

Процедури за рационално-числова аритметика
+rat, -rat, *rat, /rat, =rat, print-rat

Представяне на рационални числа чрез двойки
make-rat, numer, denom

Реализация на точковите двойки
cons, car и cdr

5. Дефиниране на съставни структури от данни

Съставните данни се задават чрез съвкупност от съответни конструктори и селектори заедно с условия (спецификации), които те трябва да удовлетворяват, за да бъде представянето коректно.

Пример. Дефинираните конструктори и селектори make-rat (без съкращаване), numer и denom удовлетворяват следните условия:

(numer (make-rat n d)) ⟶ [n],

(denom (make-rat n d)) ⟶ [d],

(make-rat (numer x) (denom x)) ⟶ [x].

5. Дефиниране на съставни структури от данни

В случая на рационалните числа тези условия се изпълняват от дефинираните по-горе процедури, защото процедурите ***car, cdr*** и ***cons*** имат аналогични свойства:

(car (cons x y)) ⟶ [x],

(cdr (cons x y)) ⟶ [y],

(cons (car z) (cdr z)) ⟶ [z].

5. Дефиниране на съставни структури от данни

Поради съображения за ефективност в езика Scheme процедурите ***car***, ***cdr*** и ***cons*** са вградени. В действителност всяка тройка от процедури, за които са верни свойствата:

(car (cons x y)) → [x]

(cdr (cons x y)) → [y]

(cons (car z) (cdr z)) → [z]

може да бъде използвана за дефиниране (реализация) на точкова двойка.

Задача за домашно упражнение 1.

Като се използва моделът на средите да се докаже, че тройката процедури: new-cons, new-car и new-cdr, дефинирани по-долу, може да се използва за конструктор и селектори на точкови двойки.

```
(define (new-cons x y)
    (define (dispatch m)
        (cond ((= m 0) x)
              ((= m 1) y)
              (else (error "Argument not 0 or 1" m))))
    dispatch)
(define (new-car z) (z 0))
(define (new-cdr z) (z 1))
```

(car (cons x y)) → [x]
(cdr (cons x y)) → [y]
(cons (car z) (cdr z)) → [z]

Задача за домашно упражнение 2.

Като се използва моделът на средите да се докаже, че тройката процедури: new-cons, new-car и new-cdr, дефинирани по-долу, може да се използва за конструктор и селектори на точкови двойки.

```
(define (new-cons x y)
    (lambda (m) (m x y)))
(define (new-car z)
    (z (lambda (p q) p)))
(define (new-cdr z)
    (z (lambda (p q) q)))
```

(car (cons x y)) → [x]
(cdr (cons x y)) → [y]
(cons (car z) (cdr z)) → [z]