


Програмування, кероване даними

Лекція 9


Визначення

Метод програмування, в якому зовнішні до програми дані використовуються з метою керування роботою програми або самі інтерпретуються в якості програми, називається програмуванням, керованим даним (англ. data driven programming).



Основою для використання такого програмування є наявність у мові програмних засобів для інтерпретації під час виконання програми зовнішніх по відношенню до неї виразів даних в якості програми.

Тобто потрібно конструювати з даних програми до їх виконання.



У функціональному програмуванні метод data driven programming легко застосовується у зв'язку з однаковою формою представлення даних і програм, можливістю обчислювати вирази, які представляють дані.



Інтерпретатор EVAL


можна безпосередньо викликати в середовищі.

У звичайному програмуванні EVAL викликати не треба, так як цей виклик неявно присутній в діалозі “програміст-система”.

Зайвий виклик інтерпретатора може зняти блокування обчислень або знайти значення від значення (подвійне обчислення).

Приклади обчислень

- `'(+ 2 3)` - ?
- `(eval '(+ 2 3))` - ?
- `(eval (cdr '(3 + 2 3)))` - ?
- `(eval x)` - ?




Функція Quote (') та інтерпретатор Eval

мають взаємно протилежні дії, анулюють
ефект одна одної.

Eval знімає блокування.

Quote блокує обчислення.



Аплікативний функціонал дозволяє викликати інші функції, застосовуючи функціональний аргумент до інших параметрів. Вони інтерпретують і перетворюють дані в програму і застосовують їх в обчисленнях.


Приклад застосування методу програмування, керованого даними

Аналітичне диференціювання алгебраїчних виразів.

Алгебраїчні вирази представляються у формі префіксних S-виразів.


У алгебраїчній вираз входять змінні, константи як операнди, знаки арифметичних операцій та назви математичних функцій (тригонометричні, логарифмічні, показникові, степеневі).

- Фрагмент програми символьного диференціювання виразу L по змінній X .



```
(define dx (lambda (l x) (cond ((atom? l)
(cond ((eq? l x) 1) (#t 0)))
(eq? (car l) '+) (list '+ (dx (car(cdr l)) x) (dx (car (cdr
(cdr l))) x)))
((eq? (car l) '*) (list '+ (list '* (dx (car(cdr l)) x) (car
(cdr (cdr l)))) (list '* (dx (car (cdr (cdr l))) x) (car
(cdr l)))))
(#t 1))))
```

- $(dx \ '(+ \ x \ (* \ 5 \ x)) \ 'x)$
- $(+ \ 1 \ (+ \ (* \ 0 \ x) \ (* \ 1 \ 5)))$



Приведений фрагмент непотужний з точки зору техніки програмування. Так як, розширення його дії на операції -, / чи інші функції вимагає зміни визначення функції dx , а саме додавання в речення `cond` нових гілок, які відповідають ситуаціям.

Було б ефективно, щоб перетворення визначалося безпосередньо на основі форми диференційованого виразу і без порівнянь.

Нова версія

```
(define dx1 (lambda (l x) (cond ((atom? l) (cond ((eq? l x) 1)
                                                    (#t 0))) (#t (apply (eval (car l)) (list (cdr l) x))))))
```

```
(define dx+ (lambda (l x) (list '+ (dx (car l) x) (dx (cadr l) x))))
```

```
(define dx* (lambda (l x) (list '+ (list '* (dx (car l) x) (cadr l))
                                     (list '* (dx (cadr l) x) (car l)))))
```

```
(define + dx+)
(define * dx*)
```

Обчислимо

- $(dxx1 \ '(+ \ x \ (* \ 5 \ x)) \ 'x)$
- $(dxx1 \ '(* \ (+ \ 4 \ x) \ 7) \ 'x)$

Відповіді системи

$(+ 1 (+ (* 0 x) (* 1 5)))$

$(+ (* (+ 0 1) 7) (* 0 (+ 4 x)))$