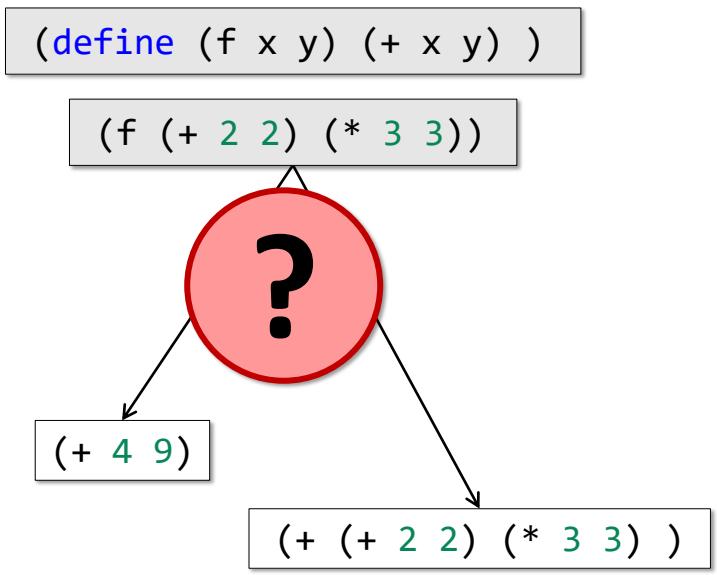
Стратегии за оценяване на изрази

доц. Атанас Семерджиев

1



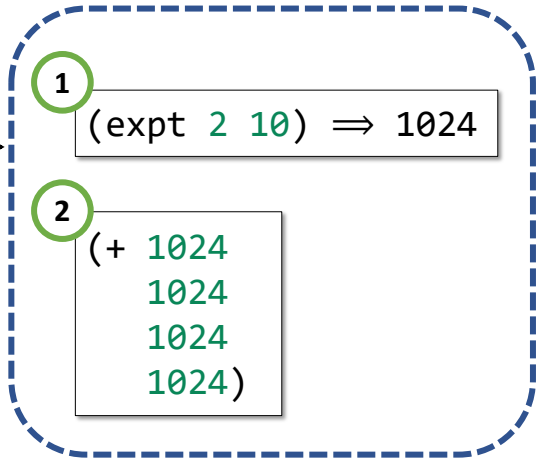
2

2

Стриктно оценяване

```
(define (f x)
   (+ x x x x))

(f (expt 2 10))
```



- *Applicative Order*
- *Call by Value*
- *Call by Reference*

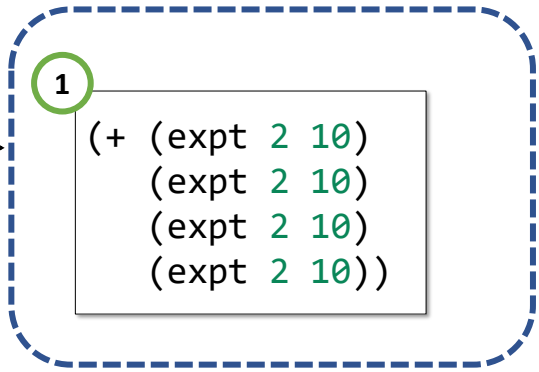
3

3

Нестриктно оценяване

```
(define (f x)
   (+ x x x x))

(f (expt 2 10))
```



- *Normal Order*
- *Call by Name*
- *Call by Need*

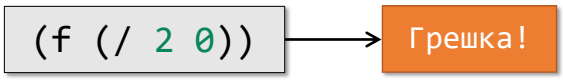
4

4

Стриктно и нестриктно оценяване

```
(define (f x) (if #f x 5) )
```

Стриктно оценяване



Нестриктно оценяване

5

5

Правилата в рамките на един език могат да бъдат разнообразни

Например:

- Операторът && в C++, Java, Haskell и т.н. може да бъде стриктен по отношение на втория си аргумент, но само ако първият му е бил истина.
- Специалните форми в Scheme следват различни правила за оценяване
- и т.н.

6

6

Апликативно оценяване

Правило:

leftmost-innermost first

```
(define (f x y)
  (+ x y))

(define (g x y)
  (* x y))
```

```
1: (f (g (+ 2 3) (+ 1 1)) (* 2 3))
2: (f (g 5 (+ 1 1)) (* 2 3))
3: (f (g 5 2) (* 2 3))
4: (f (* 5 2) (* 2 3))
5: (f 10 (* 2 3))
6: (f 10 6)
7: (+ 10 6)
8: 16
```

7

7

Call-by-value

Това е по-скоро група от стратегии за оценяване.

Аргументите се оценяват преди обръщението към функцията.

Стойността на всеки израз се пресмята и се свързва с локална за функцията променлива, като така оригиналът не се променя

8

8

Call-by-value

Забележете, че това важи за C, дори когато подавате указател.

Отново има данна, която се подава като копие – вие подавате адрес, прави се копие на този адрес и се работи с копието; указателят във функцията не е този, който сте подали).

```
void f(int* p)
{
  p = NULL;
}

void g()
{
  int var;
  int* p = &var;
  f(p);
  // p still points to var
}
```

9

9

Call-by-reference

- Това е по-скоро група от стратегии за оценяване.
- Подавате „препратка“ към оригиналното съдържание.
- точният механизъм зависи от конкретния език/имплементация и т.н. Възможно е да има различни начини за обработка дори и в рамките само на един език.
- Например в C++ reference не е просто „syntactic sugar” за указатели, а добавя цяло ново ниво, на което може да се работи с „препратки“. В някакъв смисъл можем да мислим за тях като за по-високо ниво на абстракция, което в някои частни случаи може да се имплементира чрез указатели.

10

10

Call-by-reference

```
void f(int& ref)
{
    ref = 10;
}
```

```
void g()
{
    int var = 0, t = 0;
    int& ref = var;
    std::cin >> var >> t;

    var += t;
    std::cout << var;
    ref += t;
    std::cout << var;
    f(var);
    std::cout << var;
}
```

11

11

Normal order

Правило:

leftmost-outermost first

```
(define (f x y)
  (+ x y))

(define (g x y)
  (* x y))
```

```
1: (f (g (+ 2 3) (+ 1 1)) (* 2 3))
2: (+ (g (+ 2 3) (+ 1 1)) (* 2 3))
3: (+ (* (+ 2 3) (+ 1 1)) (* 2 3))
4: (+ (* 5 (+ 1 1)) (* 2 3))
5: (+ (* 5 2) (* 2 3))
6: (+ 10 (* 2 3))
7: (+ 10 6)
8: 16
```

12

12

Call-by-name

Аргументите се замесват в тялото на функцията чрез capture-avoiding субситутуция

Предимства:

- Ако даден аргумент не се използва, автоматично се избягва неговото оценяване;
- Оценяването става точно когато има нужда.
- Може да оцени изрази, които не могат да се оценят при call-by-value

Недостатъци:

- Оценяването става точно когато има нужда.
- Може да се дублира работа.

13

13

Call-by-need

Сходен на call-by-name.

Аргументите отново се заместват в тялото на функцията чрез субституция, но се използва мемоизация.

Първият път, когато даден аргумент се оцени, стойността му се кешира. Следващите оценки използват наготово кешираната стойност.

Когато аргументите са „чисти“ (в смисъл на „pure“), семантиката е същата като на call-by-name.

14

14

Примери

C, C++, Scheme (R5RS):

- call-by-value
- Редът на оценяване на аргументите не е определен (unspecified)
- Редът на оценяване може да се избира за всяко отделно извикване. В общия случай, в рамките на една и съща програма, той може да не бъде един и същ за всички извиквания

15

15

Примери

Scheme (R5RS):

- Аргументите може да се оценяват паралелно, но семантиката трябва да е същата като за някакво последователно оценяване.

Haskell:

- call-by-need
- Странични ефекти се поддържат само чрез монади.

16

16

Асоциативност ≠ ред на оценяване

C++:

```
f() + g() + h()
```

Асоциативността на събирането е лява ⇒

```
(f() + g()) + h()
```

Редът на оценяване е неопределен (unspecified) :

възможно е `h()` да се оцени преди `f()` и `g()`

17

17

Calling convention ≠ evaluation order/strategy

C++:

```
int f(int a, int b);
```

Редът на оценяване на аргументите е unspecified; може да варира дори между различните обръщения към една и съща функция.

Конвенциите за извикване НЕ СА част нито от C, нито от C++ стандарта, но обикновено създателите на компилатора ги третират като implementation defined и респективно ги документират. Когато изберем дадена конвенция, тя се спазва стриктно.

18

18

Препоръка

Ако редът има значение, вероятно ще бъде по-добре да изнесете изразите преди обръщението към функцията!

Вместо

```
f(g(), h());
```

Използвайте

```
int arg1 = g();
int arg2 = h();
f(arg1, arg2);
```

19

19

20