Теория ФП

Лекция 2.

ВМК МГУ, 2010. Контакты: ximaera@cs.msu.su, http://haskell.secsem.ru

Семантика λ-исчисления

- Значение выражения (λx.M) – функция с одним параметром x и телом M.
  - λx.y – параметр x, тело y
  - λy.xz – параметр y, тело xz
  - λy.yyx – параметр y, тело yyx
  - λx.(λy.yyx) – параметр x, тело (λy.yyx)
    - Для простоты считаем, что эта функция имеет два параметра (x, y) и тело yyx. Поэтому можем убрать скобки: λx.(λy.yyx) <=> λxy.yyx
    - Но подставлять параметры в функцию многих аргументов можно по отдельности по мере их поступления

Семантика λ-исчисления

- Операции над функциями – редукции!
- β-редукция – подстановка терма в выражение вместо параметра
  - Основная операция λ-исчисления
- α-редукция – переименование переменной (во избежание конфликтов имен при подстановке)
  - Вспомогательная операция

Семантика λ-исчисления

- Связанные идентификаторы терма – локальные переменные функции
- Свободные идентификаторы – глобальные переменные. Говоря точнее, переменные из внешней области видимости
- Таким образом, абстрактор λ. – это ограничитель областей видимости

Моделирование

- λ-исчисление – инструмент моделирования
  - Вычислительных процессов
  - Структур данных
  - Чего только не...
- Как на основе единственной операции – β-редукции – осуществлять это моделирование?
  - Попробуем проделать это сами!

Моделирование

- λ-исчисление – инструмент моделирования
- Примеры:
  - Моделирование логических выражений

Логические выражения

- Для воспроизведения булевой алгебры необходимо определить TRUE и FALSE
  - В распоряжении – только функции 1..n аргументов, без глобальных переменных (FV = ∅)
- Пусть TRUE – функция 1 аргумента:
  - TRUE = λx.x
  - FALSE = λx.xx
  - Возможно, но сложновато

Логические выражения

- Для воспроизведения булевой алгебры необходимо определить TRUE и FALSE
  - В распоряжении – только функции 1..n аргументов, без глобальных переменных (FV = ∅)
- Пусть TRUE – функция 2 аргументов:
  - TRUE = λxy.x
  - FALSE = λxy.y
  - Попробуем создать условный оператор

Логические выражения

- Условный оператор IF имеет следующий вид:

  IF condition THEN clause_t ELSE clause_e

- Ему соответствует функция с 3 аргументами:

  IF (condition, clause_t, clause_e)

- λ-выражение для IF тоже с 3 аргументами:

  IF = λcte.X

- Каким бы взять X?

Логические выражения

- IF = λcte.X
  - IF TRUE → λte.X[c/λxy.x]
  - IF FALSE → λte.X[c/λxy.y]
- Пусть IF = λcte.cte. Тогда:
  - IF TRUE → λte.(λxy.x)te ->> λte.t
  - IF FALSE → λte.(λxy.y)te ->> λte.e
- Работает!

Логические выражения

- Пример:

  IF FALSE 1 0

  → (λcte.cte) (λxy.y) 1 0

  → (λte.(λxy.y)te) 1 0

- Дальше – два варианта. Подставляем:
  - Либо [x/t], [y/e] в λxy.y:

    → (λte.y[x/t][y/e]) 1 0

    → (λte.e) 1 0

    → **0**
  - Либо [t/1], [e/0]:

    → ((λxy.y)te)[t/1][e/0]

    → (λxy.y) 1 0

    → **0**

Редукционные стратегии

- Заметим, преобразование λ-выражения можно проводить по-разному, с разной очередностью редукций
- Может ли разная последовательность редукций одного λ-терма давать разные результаты?

Редукционные стратегии

- Возьмём терм L = (λx.xx)(λx.xx)
- Попробуем преобразовать его:

  (λx.xx)(λx.xx)

  → xx[x/λx.xx]

  → (x[x/λx.xx])(x[x/λx.xx])

  → (λx.xx)(λx.xx)

  → ...

Редукционные стратегии

- Нормальная форма λ-терма – его β-нормальная форма
  - Т. е. форма, не содержащая *β-редексов* (λx.M)N
- Но L = (λx.xx)(λx.xx) всегда будет содержать β-редекс!
- Поэтому L не имеет нормальной формы

Редукционные стратегии

- Теперь преобразуем терм IF TRUE 1 L:

1) Начнём с начала:

IF TRUE 1 L

→ (λcte.cte) (λxy.x) 1 L

→ (λte.(λxy.x)te) 1 L

→ (λte.x[x/t][y/e]) 1 L

→ (λte.t) 1 L

→ **1**

2) Начнём с конца:

IF TRUE 1 L

→ IF TRUE 1 ((λx.xx) (λx.xx))

→ IF TRUE 1 xx[x/λx.xx]

→ IF TRUE 1 ((λx.xx) (λx.xx))

→ ...

Редукционные стратегии

- Преобразовывая одно и то же выражение различным образом, мы можем как получить результат, так и «зациклиться»
- Как гарантировать получение результата?

Редукционные стратегии

- Нормальная редукционная стратегия – способ редукции λ-терма, при котором на каждом шаге редукции выбирается самый левый и самый внешний β-редекс
  - MN => сначала редуцируем M, потом N
  - (λx.M)N => сначала M[x/N], потом M и N
- Если используется η-конверсия, то η-редукции выполняются после всех β-редукций

Редукционные стратегии

- Аппликативная редукционная стратегия – способ редукции λ-терма, при котором на каждом шаге редукции выбирается β-редекс, не содержащий других β-редексов

Редукционные стратегии

- АРС работает быстрее за счёт того, что каждый β-редекс редуцируется только один раз и не дублируется:
  - Пример: (λx.xx)((λxy.x) 1 0)

    АРС:

    → (λx.xx)(1)

    → 11

    НРС:

    → ((λxy.x) 1 0)((λxy.x) 1 0)

    → (1)((λxy.x) 1 0)

    → 11
- НРС гарантирует получение нормальной формы, если нормальная форма существует

Отступление

- В λ-исчислении:
  - TRUE = λxy.x
  - FALSE = λxy.y
  - IF = λcte.cte
- В Haskell функции true, false и if можно было бы определить самостоятельно так:

```
true x y = x
false x y = y
if c t e = c t e
```

(хотя есть встроенный оператор)

Отступление

- Или, учитывая, что в Haskell есть встроенный тип Bool:

```
data Bool = True | False
```

  и что

  IF TRUE ->> λte.t

  IF FALSE ->> λte.e

  функцию if можно было бы определить так:

```
if :: Bool -> a -> a -> a
if True t e = t
if False t e = e
```

- Смысл первой строки этого определения мы разберём позднее
- Так или иначе, в C этого сделать нельзя. А почему?

Отступление

- Выражение на C

```
if (p != 0) return *p; else return 0;
```

  сработает, даже если `p == 0`

- Выражение

```
return my_if(p != 0, *p, 0);
```

  где `my_if` – функция (не макрос), при `p == 0` не сработает вне зависимости от кода `my_if`, так как все три аргумента вычисляются до вызова функции.

- В Haskell всё по-другому

Редукционные стратегии

- В императивных языках (как правило) используется аналог аппликативной РС. В Haskell – аналог нормальной.
  - Вообще говоря, существует также ленивая РС – вариация нормальной РС, позволяющая не дублировать вычисления. Об этом позже
- Оператор `if` языка С – исключение из правила, эксплуатирующее НРС. Сам по себе С не поддерживает НРС, поэтому написать свой собственный if в С нельзя
  - ...ну, разве что на макросах

Логические выражения

- Чтобы завершить ввод логических выражений в λ-исчисление, определим базовые логические операции
  - Функции TRUE и FALSE отличаются тем, что одна из них возвращает свой первый аргумент функции, а другая – второй
  - Меняя аргументы функции местами, обратим TRUE в FALSE и FALSE в TRUE
  - Поэтому NOT = λcab.cba
  - NOT TRUE = (λcab.cba) TRUE
    = (λab.(λxy.x) ba)
    = (λab.b) = FALSE

Логические выражения

- Чтобы завершить ввод логических выражений в λ-исчисление, определим базовые логические операции
  - NOT = λcab.cba
  - AND = λab.aba
  - OR = λab.aab
- Убедитесь самостоятельно: эти функции работают

Моделирование

- λ-исчисление – инструмент моделирования
- Примеры:
  - Моделирование логических выражений
  - Моделирование арифметических вычислений

Арифметика

- Нумералы Чёрча – функции двух аргументов:

  0 = FALSE = λfx.x

  1 = λfx.fx

  2 = λfx.f(fx)

  3 = λfx.f(f(fx))

  …

  n = λfx.f(...<n раз>...(fx)...)

Арифметика

- Операции над нумералами Чёрча:
  - SUCC = λnfx.f(nfx) – увеличение на единицу
  - ADD = λmnfx.mf(nfx) – сложение
  - MLT = λmnf.m(nf) – умножение
  - EXP = λmn.nm – возведение в степень

Арифметика

- Нумералы Чёрча – это натуральные числа
  - При вычитании будем считать, что (0-1)=0
- Определим *предикат* ISZERO:
  - ISZERO = λn.n (λx.FALSE) TRUE

Арифметика

- Проверим:
  - **ISZERO 0** = (λn.n (λx.FALSE) TRUE) 0

    → 0 (λx.FALSE) TRUE

    → λfx.x (λx.FALSE) TRUE

    → **TRUE**
  - **ISZERO 1** = (λn.n (λx.FALSE) TRUE) 1

    → 1 (λx.FALSE) TRUE

    → λfx.fx (λx.FALSE) TRUE

    → (λx.FALSE) TRUE

    Дважды проводим α-редукцию FALSE во избежание конфликта имен: вначале [x/a][y/b], потом обратно

    → (λx.λab.b) TRUE → λab.b → **FALSE**

Арифметика

- С помощью ISZERO запишем:
  - PRED = λn.n (λgk.ISZERO (g 1) k (ADD (g k) 1)) (λv.0) 0
  - SUB = λmn.n PRED m
- Легко убедиться, что, например, выражения

  5

  и

  ADD (MULT 2 1) (SUCC 2)

  эквивалентны.

Моделирование

- λ-исчисление – инструмент моделирования
- Примеры:
  - Моделирование логических выражений
  - Моделирование арифметических вычислений
  - Моделирование структур данных

Структуры данных

- Простейшая структура данных – пара из двух элементов
- В синтаксисе C:

```
struct pair {
    TYPE_1 first;
    TYPE_2 second;
};
```

- В терминологии C++:

```
template <typename _T1, typename _T2>
struct std::pair;
```

Структуры данных

- Пара на языке λ-исчисления:

  PAIR = λxyf.fxy

  FIRST = λp.p TRUE

  SND = λp.p FALSE

- Вдумайтесь:

  PAIR 1 2 = (λf.f 1 2) – функция, принимающая в качестве аргумента функцию, которая производит действия с элементами пары

Структуры данных

- Иллюстрация: SND (PAIR 2 3)

  → SND ((λxyf.fxy) 2 3)

  → SND ((λyf.f 2 y) 3)

  → SND (λf.f 2 3)

  → (λp.p FALSE) (λf.f 2 3)

  → (λf.f 2 3) FALSE

  → FALSE 2 3

  → (λxy.y) 2 3

  → 3

Структуры данных

- Обратите внимание на функцию FALSE!

  → SND ((λxyf.fxy) 2 3)

  → SND ((λyf.f 2 y) 3)

  → SND (λf.f 2 3) – FALSE – ещё элемент SND

  → (λp.p FALSE) (λf.f 2 3)

  → (λf.f 2 3) FALSE – FALSE – уже аргумент!

  → FALSE 2 3 – функция FALSE вызывается!

  → (λxy.y) 2 3

  → 3

Структуры данных

- Аналогично можно определять другие структуры, скажем, IP-адрес:

  IP = λabcdf.fabcd

  IP_OCTET_A = λp.p (λabcd.a)

  IP_OCTET_B = λp.p (λabcd.b)

  IP_OCTET_C = λp.p (λabcd.c)

  IP_OCTET_D = λp.p (λabcd.d)

- Но структура PAIR интересна ещё и потому, что через неё удобно определять список

Списки Lisp

- Связные списки Lisp – это набор пар вида (данные, ссылка на следующую пару).
- Например, список (3 4 5) представляется как (3, →(4, →(5, →nil)))
- В диалектах Lisp есть следующие функции:
  - cons – конструктор пары
  - car – первый элемент пары
  - cdr – второй элемент пары
  - nil – пустой список
  - null? – проверка на пустой список

Списки Lisp

- Например:
  - `(cons 2 3)` → `'(2 3)`
  - `(cons 3 (cons 4 5))` → `'(3 4 5)`
  - `(car '(3 4 5))` → `3`
  - `(cdr '(3 4 5))` → `'(4 5)`
  - `(cdr (cdr '(3 4 5)))` → `'(5)`
  - `(null? (cdr (cdr (cdr '(3 4 5)))))`
    → `#t` (true)

Структуры данных

- Опишем на языке λ-исчисления список Lisp
  - Для разграничения пар используем дополнительные пары с первым элементом, равным FALSE
  - CONS = λxy.(PAIR FALSE (PAIR x y))
  - CAR = λx.(FIRST (SND x))
  - CDR = λx.(SND (SND x))
  - NIL = λx.x
  - ISNULL = FIRST

Моделирование

- λ-исчисление – инструмент моделирования
- Примеры:
  - Моделирование логических выражений
  - Моделирование арифметических вычислений
  - Моделирование структур данных
  - Моделирование «циклов» (итеративных процессов)

Итерации

- Цикл в понимании языка C на λ-исчислении реализовать не удастся
  - for, while, do...while – все такие циклы требуют изменяемых переменных (счётчиков цикла или изменяющихся условий), а в λ-исчислении есть только константы
- Однако возможно реализовать цикл в более широком понимании этого слова – итеративный процесс
  - Рекурсия – тоже цикл!

Итерации

- Неподвижная точка (fixed point) функции y=f(x) – такая $x \in D(f)$, что $f(x) = x$
  - Для $f(x) = x^2$ – точки 0 и 1
  - Для $f(x) = x$ – вся область определения
- Функция (или *комбинатор*) неподвижной точки – функция, вычисляющая для функции её неподвижную точку
  - Традиционно обозначается буквой Y
  - Известна также как Y Combinator

Итерации

- Основное свойство комбинатора неподвижной точки можно записать так:

  $a \in \Lambda: Y\ a = a\ (Y\ a)$

- Наиболее известные примеры комбинатора Y:
  - Хаскелл Карри:

    $Y_K = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$

  - Алан Тьюринг:

    $Y_T = (\lambda xy.(y(xxy)))(\lambda xy.(y(xxy)))$

Итерации

- Как использовать Y для создания рекурсии?
- Пример: факториал числа n
  - Запишем функцию, реализующую одну итерацию:

    IFACT = λfn.IF (ISZERO n) 1 (MLT n (f (PRED n)))
  - Запишем рекурсивную функцию:

    FACT = Y IFACT

Итерации

- Как использовать Y для создания рекурсии?
- Пример: факториал числа n
  - **FACT 3** → (Y IFACT) 3 → IFACT (Y IFACT) 3 → IFACT FACT 3
    → (λfn.IF (ISZERO n) 1 (MLT n (f (PRED n)))) FACT 3
    → IF (ISZERO 3) 1 (MLT 3 (FACT (PRED 3)))
    → (MLT 3 (FACT 2))
    → (MLT 3 (IFACT FACT 2))
    → (MLT 3 (IF (ISZERO 2) 1 (MLT 2 (FACT (PRED 2)))))
    → (MLT 3 (MLT 2 (FACT (PRED 2))))
    → (MLT 3 (MLT 2 (FACT 1)))
    → (MLT 3 (MLT 2 (MLT 1 (FACT 0))))
    → (MLT 3 (MLT 2 (MLT 1 (IFACT FACT 0))))
    → (MLT 3 (MLT 2 (MLT 1 (IF (ISZERO 0) 1 (MLT 0 (FACT (PRED 0)))))))
    → (MLT 3 (MLT 2 (MLT 1 1))) → (MLT 3 (MLT 2 1)) → (MLT 3 2) → **6**

Итерации

- Функциональные языки основаны на λ-исчислении
- Поэтому в функциональном программировании единственным средством организации циклических вычислений также является рекурсия
- Для нейтрализации побочных эффектов рекурсии существует хвостовая рекурсия
- Но об этом в другой раз

Организационное замечание

- С 12 октября занятия проходят в аудитории П-14. Время то же: вторник, 18:00.