Язык ML: обобщённые типы, функциональное и объектно-ориентированное программирование

Фатахов Тимур, Буслама Анис Группа: 5030102/20201

«Обобщённые типы (Generics) в ML»

ML поддерживает полиморфные (обобщённые) типы:
– 'a list, 'a option, 'a tree и т.п

Переменные типов ('a, 'b) позволяют писать один алгоритм для разных типов данных

Примеры полиморфных функций:

- length : 'a list -> int
- map : ('a -> 'b) -> 'a list -> 'b list

Плюсы:

- нет дублирования кода под каждый тип
- строгая статическая типизация
- без динамического «кастования»

```
1  (* Примеры обобщённых типов *)
2  val emptyList  : 'a list = [];
3  val maybeValue : 'a option = NONE;
4
5  (* Пример полиморфной функции *)
6  fun length xs =
7    let
8      fun aux (n, [])       = n
9        | aux (n, _ :: tl) = aux (n + 1, tl)
10   in
11     aux (0, xs)
12   end;
13
```

```
Output
> val emptyList = []: ∀ 'a . 'a list;
> val maybeValue = NONE: ∀ 'a . 'a option;
> val length = fn: ∀ 'a . 'a list → int;
```

Пример: одна функция — разные типы

ML поддерживает полиморфные (обобщённые) типы:

- 'a list, 'a option, 'a tree и т.п.

Переменные типов ('a, 'b) позволяют писать один алгоритм для разных типов данных

Примеры полиморфных функций:

- length : 'a list -> int
- map : ('a -> 'b) -> 'a list -> 'b list

Преимущества:

- нет дублирования кода под каждый конкретный тип
- строгая статическая типизация
- компилятор **сам выводит** обобщённые типы по коду функции

```
1  fun id x = x;
2
3  fun listLength xs =
4    let
5      fun aux (acc, [])      = acc
6        | aux (acc, _ :: tl) = aux (acc + 1, tl)
7    in
8      aux (0, xs)
9    end;
10
11 val _ =
12   let
13     val ints    = [1, 2, 3]
14     val strings = ["a", "b", "c", "d"]
15   in
16     print ("len [1,2,3] = " ^
17            Int.toString (listLength ints) ^ "\n");
18     print ("len [\"a\",\"b\",\"c\",\"d\"] = " ^
19            Int.toString (listLength strings) ^ "\n");
20     print ("id 42 = " ^ Int.toString (id 42) ^ "\n");
21     print ("id \"hello\" = " ^ id "hello" ^ "\n")
22   end;
23
```

Output

```
> val id = fn: ∀ 'a . 'a → 'a;
> val listLength = fn: ∀ 'a . 'a list → int;
Printed: len [1,2,3] = 3
Printed: len ["a","b","c","d"] = 4
Printed: id 42 = 42
Printed: id "hello" = hello
```

Функциональный стиль в ML

Функции — значения первого класса:

- можно передавать как аргументы
- возвращать из других функций
- хранить в структурах данных

Неизменяемые структуры данных по умолчанию

Широкое использование:

- рекурсии
- паттерн-матчинга

Функции map и filter — классические примеры:

- одна реализация для любых типов элементов списка

```
1  fun rev xs =
2    let
3      fun loop ([], acc)      = acc
4        | loop (y :: ys, acc) = loop (ys, y :: acc)
5    in
6      loop (xs, [])
7    end;
8
9  fun map f xs =
10   let
11     fun loop ([], acc)      = rev acc
12       | loop (y :: ys, acc) = loop (ys, f y :: acc)
13   in
14     loop (xs, [])
15   end;
```

Output

```
> val rev = fn: ∀ 'a . 'a list → 'a list;
> val map = fn: ∀ 'a 'b . ('a → 'b) → 'a list → 'b list;
```

```
fun filter p xs =
  let
    fun loop ([], acc) = rev acc
      | loop (y :: ys, acc) =
          if p y
          then loop (ys, y :: acc)
          else loop (ys, acc)
  in
    loop (xs, [])
  end;

fun showIntList xs =
  let
    fun loop [] acc         = acc ^ "]"
      | loop [x] acc        = acc ^ Int.toString x ^ "]"
      | loop (x :: xs) acc = loop xs (acc ^ Int.toString x ^ "; ")
  in
    loop xs "["
  end;

val _ =
  let
    val nums    = [1, 2, 3, 4, 5]
    val squares = map (fn x => x * x) nums
    val evens   = filter (fn x => x mod 2 = 0) nums
  in
    print ("nums    = " ^ showIntList nums ^ "\n");
    print ("squares = " ^ showIntList squares ^ "\n");
    print ("evens   = " ^ showIntList evens ^ "\n")
  end;
```

Output

```
> val filter = fn: ∀ 'a . ('a → bool) → 'a list → 'a list;
> val showIntList = fn: int list → string;
Printed: nums    = [1; 2; 3; 4; 5]
Printed: squares = [1; 4; 9; 16; 25]
Printed: evens   = [2; 4]
```

Ввод-вывод в ML: стандартная библиотека

Standard ML имеет Basis Library:

- модули для типов, строк, списков, массивов и т.д.

Для текстового ввода-вывода:

- структура TextIO
- функции openIn, openOut, inputLine, output, closeIn, closeOut

Для бинарного ввода-вывода:

- структура BinIO

Для работы с ОС:

- структуры OS, Unix

Этого достаточно для:

- чтения/записи файлов
- работы с потоками стандартного ввода/вывода

```
1 val openIn    : string -> TextIO.instream
2 val inputLine : TextIO.instream -> string option
3 val closeIn   : TextIO.instream -> unit
```

Пример: подсчёт строк в файле

Программа:

- спрашивает имя файла у пользователя
- открывает файл через TextIO.openIn
- считает количество строк до NONE
- закрывает поток через TextIO.closeIn

Используется:

- тип string option и паттерн-матчинг
- цикл через рекурсивную функцию loop

```
1  fun countLines ins =
2    let
3      fun loop n =
4        case TextIO.inputLine ins of
5             SOME _ => loop (n + 1)
6           | NONE   => n
7    in
8      loop 0
9    end
10
11 val _ =
12   let
13     val () = print "Введите имя файла и нажмите Enter:\n"
14   in
15     case TextIO.inputLine TextIO.stdIn of
16          SOME name =>
17            let
18              val name = String.extract (name, 0, SOME (size name - 1))
19              val ins = TextIO.openIn name
20              val lines = countLines ins
21              val () = TextIO.closeIn ins
22            in
23              print ("В файле \"" ^ name ^ "\" "
24                     ^ Int.toString lines ^ " строк(и)\n")
25            end
26        | NONE =>
27            print "Не удалось прочитать имя файла\n"
28   end
29
```

Конкурентность и параллелизм в семействе ML

Базовый Standard ML — последовательный, но существуют расширения:

- Concurrent ML (CML) — расширение ML для конкурентного программирования

Основные идеи CML:

- потоки (threads)
- каналы сообщений (channel)
- операции send / recv
- синхронизация через события и функцию sync

ML-подход:

- конкурентность описывается через функциональные абстракции
- события и каналы — обычные значения, их можно комбинировать и передавать в функции

```
1  signature CML =
2  sig
3    type 'a chan
4    val channel : unit -> 'a chan
5    val send    : 'a chan * 'a -> unit
6    val recv    : 'a chan -> 'a
7  end
```

Пример concurrent-кода: два потока и логгер

Создаём канал ch для строк

Два рабочих потока:

- печатают шаг
- отправляют сообщение в канал

Поток-логгер:

- читает из канала
- печатает «Лог: ...»

Запуск через RunCML.doit

```
1  structure CML =
2  struct
3    datatype 'a chan = Chan of 'a list ref
4
5    fun channel () = Chan (ref [])
6
7    fun send (Chan r, x) =
8      r := !r @ [x]
9
10   fun recv (Chan r) =
11     case !r of
12         [] => "EMPTY\n"
13       | x::xs => (r := xs; x)
14 end
15
```

Output

```
> structure CML = struct
    val Chan = Chan: ∀ 'a . 'a list ref → 'a chan;
    val channel = fn: ∀ 'a . unit → 'a chan;
    val send = fn: ∀ 'a . 'a chan * 'a → unit;
    val recv = fn: string chan → string;
    datatype 'a chan = {
      con Chan = Chan: ∀ 'a . 'a list ref → 'a chan;
    };
  end;
```

```
structure Example =
struct
  open CML

  fun worker name ch =
    let
      fun loop 0 = ()
        | loop n =
            ( print (name ^ ": шаг " ^ Int.toString n ^ "\n");
              send (ch, name ^ " готов\n");
              loop (n - 1) )
    in
      loop 3
    end

  fun logger ch =
    let
      fun loop 0 = ()
        | loop n =
            let
              val msg = recv ch
            in
              print ("Лог: " ^ msg);
              loop (n - 1)
            end
    in
      loop 6   (* два воркера × 3 шага *)
    end

  fun main () =
    let
      val ch = channel ()
      val _  = worker "Поток 1" ch
      val _  = worker "Поток 2" ch
      val _  = logger ch
    in
      ()
    end
end;

Example.main ();
```

```
> structure Example = struct
    val Chan = Chan: ∀ 'a . 'a list ref → 'a chan;
    val channel = fn: ∀ 'a . unit → 'a chan;
    val send = fn: ∀ 'a . 'a chan * 'a → unit;
    val recv = fn: string chan → string;
    val worker = fn: string → string chan → unit;
    val logger = fn: string chan → unit;
    val main = fn: unit → unit;
    datatype 'a chan = {
      con Chan = Chan: ∀ 'a . 'a list ref → 'a chan;
    };
  end;
> val it = (): unit;
Printed: Поток 1: шаг 3
Printed: Поток 1: шаг 2
Printed: Поток 1: шаг 1
Printed: Поток 2: шаг 3
Printed: Поток 2: шаг 2
Printed: Поток 2: шаг 1
Printed: Лог: Поток 1 готов
Printed: Лог: Поток 1 готов
Printed: Лог: Поток 1 готов
Printed: Лог: Поток 2 готов
Printed: Лог: Поток 2 готов
Printed: Лог: Поток 2 готов
```

Объектно-ориентированный подход в ML

В ML нет отдельного ключевого слова class

Объект можно представить как запись с полями и функциями:

- данные → поля
- поведение → функции

Инкапсуляция достигается через:

- абстрактные типы и модульную систему

Полиморфизм и композиция хорошо сочетаются с функциональным стилем

```
 4 type shape =
 5   { name : string
 6   , area : unit -> int
 7   }
 8
 9 fun makeCircle r =
10   { name = "Круг"
11   , area = (fn () => r * r)
12   }
13
14 fun makeRectangle (w, h) =
15   { name = "Прямоугольник"
16   , area = (fn () => w * h)
17   }
18
19 fun describe (s : shape) =
20   #name s ^ " с площадью " ^
21   Int.toString ((#area s) ())
```

Пример: работа с фигурами как с объектами

makeCircle и makeRectangle — конструкторы объектов

describe работает с любым shape, не зная внутренней реализации

Создаём список фигур и печатаем их описание

```
type shape =
  { name : string
  , area : unit -> int
  }

fun makeCircle r =
  { name = "Круг"
  , area = (fn () => r * r)
  }

fun makeRectangle (w, h) =
  { name = "Прямоугольник"
  , area = (fn () => w * h)
  }

fun describe (s : shape) =
  #name s ^ " с площадью " ^
  Int.toString ((#area s) ())

val _ =
  let
    val shapes : shape list =
      [ makeCircle 3
      , makeRectangle (2, 3)
      ]
  in
    List.app (fn s =>
      print (describe s ^ "\n")) shapes
  end
```

Выводы

- Языки семейства ML опираются на строгую статическую типизацию с выводом типов
- Обобщённые типы позволяют писать один алгоритм для разных структур данных
- Функциональный стиль (рекурсия, map, filter, неизменяемые данные) делает код компактным и предсказуемым
- Стандартные средства ввода-вывода (на базе TextIO) позволяют работать с файлами и потоками данных
- Конкурентность в ML реализуется через расширения (например, Concurrent ML) на основе потоков и каналов сообщений
- Объектный подход можно выразить через записи и функции, не вводя отдельный синтаксис классов
- Все разобранные примеры кода собраны в каталоге 06-ml/05/src и связаны с темами презентации