~ *Методы Вычислений* ~

Автор: Пицик Харитон

Лекция 16 сентября 2025

Основные типы данных

`unit` — (). Аналог `void` из `C++`.

```
- ();
val it = () : unit
```

Общение с интерпретатором — он может завести временную переменную `it`, к которой можно даже обратиться

```
- 2 + 3;
...
- it;
val it = 5 : int
```

Знак “;” сигнализирует о завершении команды.

Для работы с, например, списками, нам по большому счёту неважно, с каким типом мы работаем (главное, что список):

```
fun f (l : 'a list) : 'a = hd (tl (tl l))
```

```
fun g (l : 'a list, l` : 'a list) : 'a * 'a =
  (hd l, hd l')
...
g ([1, 2], [5, 6]) => (1, 5)
g ([1.2, 3.7], [4.3, 8.1]) => (1.2, 4.3)
```

т.е. она работает с одинаковыми типами. Исправим:

```
fun g(l : 'a list, l` : 'b list) : 'a * 'b =
  (hd l, hd l')
```

Это — полиморфные функции (может принимать различные типы данных), а сам факт их присутствия — полиморфизм. f, g — полиморфные функции, `sml` полиморфен по типам данных.

```
fun f'' (l : ''a list) : ''a list =
  if null l then []
  else if null (tl l) then []
  else if hd l = hd (tl l) then f''(tl l)
  else hd l :: f''(tl l)
```

`''a` — тип, который поддерживает сравнение на равенство.

у нас не будет ооп.

Задача переворота списка

Реализуем функцию, которая будет из списка $[1, 2, 3, 4, 5]$ делать $[5, 4, 3, 2, 1]$.

!!!ПЛОХО!!!

```
fun reverse (l : 'a list) : 'a list =
  if null l then l
  else reverse (tl l) @ [hd l]
```

Лучше так:

```
fun reverse (l : 'a list) : 'a list =
  let
    fun revHelper (l : 'a list, acc : 'a list) : 'a list =
      if null l then acc
      else revHelper (tl l, hd l :: acc)
  in
    revHelper (l, [])
  end
```

- [1, 2, 3]
- reverse [1, 2, 3]
- revHelper ([1, 2, 3], [])
- revHelper ([2, 3], [1])
- revHelper ([3], [2, 1])
- revHelper ([], [3, 2, 1])
- [3, 2, 1]

Задача поиска

```
fun search (l : 'a list, f : 'a -> bool) : 'a =
  if f (hd l) then hd l
  else search (tl l, f)
```

Но что если список вообще пустой, или там нет искомого? Здесь на помощь приходит `option` — специальный тип данных.

```
int option => NONE отсутствие
int option => SOME <значение>
```

```
SOME 15 : int option
```

```
fun search (l : 'a list, f : 'a -> bool) : 'a option =
  if null l then NONE
  else
    let
      val b = hd l
    in
      if f b then SOME b
      else search (tl l, f)
    end
```

Если дано:

```
a : 'a option
```

Можно проверить, есть ли в нём что-то:

```
isSome a => true (SOME) / false (NONE)
```

А чтобы извлечь:

```
valOf : 'a option -> 'a.
```

```
fun max (l : 'a list, f : 'a * 'a -> bool) : 'a option = (* т.к. может быть пустой *)
  if null then NONE
  else
    let
      fun maxNonEmpty (l : 'a list) : 'a =
        if null (tl l) then hd l
        else
```

```
        let
          val m = maxNonEmpty (tl l)
        in
          if f (hd l, m) then m
          else hd l
        end
    in
      SOME (maxNonEmpty l)
    end
```

- max([5, 3, 7, 4], op <)
- SOME(maxNonEmpty [5, 3, 7, 4])
- Вход в рекурсию
- m = maxNonEmpty [3, 7, 4]
- m = maxNonEmpty [7, 4]
- m = maxNonEmpty [4]
- Выход из рекурсии
- 4
- 7
- 7
- 7
- SOME 7