Лекция 6

Абстракция структурной рекурсии

Функциональное и логическое программирование

КамчатГТУ, 2013 г.

1. Обработка списков
   - Мотивационный пример
   - Вывод рекурсивных уравнений
2. Функция свёртки
   - Абстракция цикла по списку
   - Полезные операторы
3. Обобщение свёртки
   - Катаморфизм
   - Свёртка для потоков
   - Свёртка деревьев

1. Обработка списков
   - Мотивационный пример
   - Вывод рекурсивных уравнений
2. Функция свёртки
   - Абстракция цикла по списку
   - Полезные операторы
3. Обобщение свёртки
   - Катаморфизм
   - Свёртка для потоков
   - Свёртка деревьев

Обработка текстового файла

Задача:

Имеется текстовый файл с программой на простом алгоритмическом языке.

```
x := 4
y := 1
if x < 0 then 6
y := x * y
x := x - a
goto 2
stop y
```

Необходимо получить список объявленных переменных, используемых в программе.

Обработка текстового файла

Задача:

Имеется текстовый файл с программой на простом алгоритмическом языке.

```
x := 4
y := 1
if x < 0 then 6
y := x * y
x := x - a
goto 2
stop y
```

Необходимо получить список объявленных переменных, используемых в программе.

Императивное решение:

1. Завести пустой список (массив) res.

Обработка текстового файла

Задача:

Имеется текстовый файл с программой на простом алгоритмическом языке.

```
x := 4
y := 1
if x < 0 then 6
y := x * y
x := x - a
goto 2
stop y
```

Необходимо получить список объявленных переменных, используемых в программе.

Императивное решение:

1. Завести пустой список (массив) res.
2. Пока не окончится файл $file$

Обработка текстового файла

Задача:

Имеется текстовый файл с программой на простом алгоритмическом языке.

```
x := 4
y := 1
if x < 0 then 6
y := x * y
x := x - a
goto 2
stop y
```

Необходимо получить список объявленных переменных, используемых в программе.

Императивное решение:

1. Завести пустой список (массив) res.
2. Пока не окончится файл $file$
   2.1 читать строку s из файла,

Обработка текстового файла

Задача:

Имеется текстовый файл с программой на простом алгоритмическом языке.

```
x := 4
y := 1
if x < 0 then 6
y := x * y
x := x - a
goto 2
stop y
```

Необходимо получить список объявленных переменных, используемых в программе.

Императивное решение:

1. Завести пустой список (массив) res.
2. Пока не окончится файл $file$
   - 2.1 читать строку s из файла,
   - 2.2 разделить строку s на список слов w.

Обработка текстового файла

Задача:

Имеется текстовый файл с программой на простом алгоритмическом языке.

```
x := 4
y := 1
if x < 0 then 6
y := x * y
x := x - a
goto 2
stop y
```

Необходимо получить список объявленных переменных, используемых в программе.

Императивное решение:

1. Завести пустой список (массив) res.
2. Пока не окончится файл $file$
   - 2.1 читать строку s из файла,
   - 2.2 разделить строку s на список слов w.
   - 2.3 Если $w[2] = ':='$, то

Обработка текстового файла

Задача:

Имеется текстовый файл с программой на простом алгоритмическом языке.

```
x := 4
y := 1
if x < 0 then 6
y := x * y
x := x - a
goto 2
stop y
```

Необходимо получить список объявленных переменных, используемых в программе.

Императивное решение:

1. Завести пустой список (массив) res.
2. Пока не окончится файл $file$
   - 2.1 читать строку s из файла,
   - 2.2 разделить строку s на список слов w.
   - 2.3 Если $w[2] =$':=', то
     - 3.2.1 если $w[1]$ еще не записано в res, то добавить его в res.

Обработка текстового файла

Задача:

Имеется текстовый файл с программой на простом алгоритмическом языке.

```
x := 4
y := 1
if x < 0 then 6
y := x * y
x := x - a
goto 2
stop y
```

Необходимо получить список объявленных переменных, используемых в программе.

Императивное решение:

1. Завести пустой список (массив) res.
2. Пока не окончится файл $file$
   - 2.1 читать строку s из файла,
   - 2.2 разделить строку s на список слов w.
   - 2.3 Если $w[2] =$':=', то
     - 3.2.1 если $w[1]$ еще не записано в res, то добавить его в res.
3. Конец цикла.

Обработка текстового файла

Задача:

Имеется текстовый файл с программой на простом алгоритмическом языке.

```
x := 4
y := 1
if x < 0 then 6
y := x * y
x := x - a
goto 2
stop y
```

Необходимо получить список объявленных переменных, используемых в программе.

Императивное решение:

1. Завести пустой список (массив) res.
2. Пока не окончится файл $file$
   - 2.1 читать строку s из файла,
   - 2.2 разделить строку s на список слов w.
   - 2.3 Если $w[2] =$':=', то
     - 3.2.1 если $w[1]$ еще не записано в res, то добавить его в res.
3. Конец цикла.
4. Вернуть res.

Обработка текстового файла

Задача:

Имеется текстовый файл с программой на простом алгоритмическом языке.

```
x := 4
y := 1
if x < 0 then 6
y := x * y
x := x - a
goto 2
stop y
```

Необходимо получить список объявленных переменных, используемых в программе.

Императивное решение:

1. Завести пустой список (массив) res.
2. Пока не окончится файл $file$
   - 2.1 читать строку s из файла,
   - 2.2 разделить строку s на список слов w.
   - 2.3 Если $w[2] =$':=', то
     - 3.2.1 если $w[1]$ еще не записано в res, то добавить его в res.
3. Конец цикла.
4. Вернуть res.

Обработка текстового файла

Задача:

Имеется текстовый файл с программой на простом алгоритмическом языке.

```
x := 4
y := 1
if x < 0 then 6
y := x * y
x := x - a
goto 2
stop y
```

Необходимо получить список объявленных переменных, используемых в программе.

Императивное решение:

1. Завести пустой список (массив) res.
2. Пока не окончится файл $file$
   - 2.1 читать строку s из файла,
   - 2.2 разделить строку s на список слов w.
   - 2.3 Если $w[2] = ':='$, то
     - 3.2.1 если $w[1]$ еще не записано в res, то добавить его в res.
3. Конец цикла.
4. Вернуть res.

Декларативное определение

Список переменных — это множество первых элементов строк с присвоением, полученных из файла и разбитых на слова.

Функциональное решение

Определение

```
variables =
  ;множество
   ;первых элементов
    ;в строках с присвоением,
     ;разбитых на слова
      ;и полученных из файла
```

Функциональное решение

Определение

```
variables =
  ;множество
   ;первых элементов
    ;в строках с присвоением,
     ;разбитых на слова
      ;и полученных из файла
```

Программа

```
(define (variables file)
  (;множество
   (;первых элементов
    (;в строках с присвоением,
     (;разбитых на слова
      (;и полученных из файла file))))))
```

Функциональное решение

Определение

```
variables =
  ;множество
   ;первых элементов
    ;в строках с присвоением,
     ;разбитых на слова
      ;и полученных из файла
```

Программа

```
(define (variables file)
  (set
   (;первых элементов
    (;в строках с присвоением,
     (;разбитых на слова
      (;полученных из файла file))))))
```

Функциональное решение

Определение

```
variables =
  ;множество
   ;первых элементов
    ;в строках с присвоением,
     ;разбитых на слова
      ;и полученных из файла
```

Программа

```
(define (variables file)
  (set
   (map first
    (;в строках с присвоением,
     (;разбитых на слова
      (;полученных из файла file))))))
```

Функциональное решение

Определение

```
variables =
  ;множество
   ;первых элементов
    ;в строках с присвоением,
     ;разбитых на слова
      ;и полученных из файла
```

Программа

```
(define (variables file)
  (set
   (map first
    (filter assignment?
     (;разбитых на слова
      (;полученных из файла file))))))
```

Функциональное решение

Определение

```
variables =
  ;множество
   ;первых элементов
    ;в строках с присвоением,
     ;разбитых на слова
      ;и полученных из файла
```

Программа

```
(define (variables file)
  (set
   (map first
    (filter assignment?
     (map string->words
      (;полученных из файла file))))))
```

Функциональное решение

Определение

```
variables =
  ;множество
   ;первых элементов
    ;в строках с присвоением,
     ;разбитых на слова
      ;и полученных из файла
```

Программа

```
(define (variables file)
  (set
   (map first
    (filter assignment?
     (map string->words
      (file->lines file))))))
```

Функциональное решение

Определение

```
variables =
  ;множество
   ;первых элементов
    ;в строках с присвоением,
     ;разбитых на слова
      ;и полученных из файла

assignment? =
  ;равен ':=' второй элемент
```

Программа

```
(define (variables file)
  (set
   (map first
    (filter assignment?
     (map string->words
      (file->lines file))))))

(define (assignment? l)
  ;равен ':=' второй элемент l)
```

Функциональное решение

Определение

```
variables =
  ;множество
   ;первых элементов
    ;в строках с присвоением,
     ;разбитых на слова
      ;и полученных из файла

assignment? =
  ;равен ':=' второй элемент
```

Программа

```
(define (variables file)
  (set
   (map first
    (filter assignment?
     (map string->words
      (file->lines file))))))

(define (assignment? l)
  (eq? ':= ;второй элемент l))
```

Функциональное решение

Определение

```
variables =
  ;множество
   ;первых элементов
    ;в строках с присвоением,
     ;разбитых на слова
      ;и полученных из файла

assignment? =
  ;равен ':=' второй элемент
```

Программа

```
(define (variables file)
  (set
   (map first
    (filter assignment?
     (map string->words
      (file->lines file))))))

(define (assignment? l)
  (eq? ':= (second l)))
```

Функциональное решение

Определение

```
variables =
  ;множество
   ;первых элементов
    ;в строках с присвоением,
     ;разбитых на слова
      ;и полученных из файла

assignment? =
  ;равен ':=' второй элемент
```

Программа

```
(define (variables file)
  ((o set
      (map first)
      (filter assignment?)
      (map string->words)
      file->lines) file))

(define (assignment? l)
  (eq? ':= (second l)))
```

Функциональное решение

Определение

```
variables =
  ;множество
   ;первых элементов
    ;в строках с присвоением,
     ;разбитых на слова
      ;и полученных из файла

assignment? =
  ;равен ':=' второй элемент
```

Программа

```
(define (variables file)
  ((o set
      (map first)
      (filter assignment?)
      (map string->words)
      file->lines) file))

(define (assignment? str)
  ((o (eq? ':=) second) str))
```

Функциональное решение

Определение

```
variables =
  ;множество
   ;первых элементов
    ;в строках с присвоением,
     ;разбитых на слова
      ;и полученных из файла

assignment? =
  ;равен ':=' второй элемент
```

Программа

```
(define variables
  (o set
     (map first)
     (filter assignment?)
     (map string->words)
     file->lines))

(define assignment?
  (o (eq? ':=) second))
```

Функциональное решение

```
(define variables
  (o set
     (map first)
     (filter assignment?)
     (map string->words)
     file->lines))
```

Функциональное решение

```
(define variables
  (o set
     (map first)
     (filter assignment?)
     (map string->words)
     file->lines))
```

```
["x := 4"
 "y := 1"
 "if x < 0 then 6"
 "y := x * y"
 "x := x - a"
 "goto 2"
 "stop y"]
```

1. читаем список строк из файла,

Функциональное решение

```
(define variables
  (o set
     (map first)
     (filter assignment?)
     (map string->words)
     file->lines))
```

```
["x := 4"
 "y := 1"
 "if x < 0 then 6"
 "y := x * y"
 "x := x - a"
 "goto 2"
 "stop y"]
```

1. читаем список строк из файла,
2. в полученном списке разбиваем все элементы на слова,

Функциональное решение

```
(define variables
  (o set
     (map first)
     (filter assignment?)
     (map string->words)
     file->lines))
```

```
[[x := 4]
 [y := 1]
 [if x < 0 then 6]
 [y := x * y]
 [x := x - a]
 [goto 2]
 [stop y]]
```

1. читаем список строк из файла,
2. в полученном списке разбиваем все элементы на слова,

Функциональное решение

```
(define variables
  (o set
     (map first)
     (filter assignment?)
     (map string->words)
     file->lines))
```

```
[[x := 4]
 [y := 1]
 [if x < 0 then 6]
 [y := x * y]
 [x := x - a]
 [goto 2]
 [stop y]]
```

1. читаем список строк из файла,
2. в полученном списке разбиваем все элементы на слова,
3. – выбираем только те элементы, у которых второе слово ':=',

Функциональное решение

```
(define variables
  (o set
     (map first)
     (filter assignment?)
     (map string->words)
     file->lines))
```

```
[[x := 4]
 [y := 1]
 [if x < 0 then 6]
 [y := x * y]
 [x := x - a]
 [goto 2]
 [stop y]]
```

1. читаем список строк из файла,
2. в полученном списке разбиваем все элементы на слова,
3. – выбираем только те элементы, у которых второе слово ':=',

Функциональное решение

```
(define variables
  (o set
     (map first)
     (filter assignment?)
     (map string->words)
     file->lines))
```

```
[[x := 4]
 [y := 1]
 [y := x * y]
 [x := x - a]]
```

1. читаем список строк из файла,
2. в полученном списке разбиваем все элементы на слова,
3. – выбираем только те элементы, у которых второе слово ':=',

Функциональное решение

```
(define variables
  (o set
     (map first)
     (filter assignment?)
     (map string->words)
     file->lines))
```

```
[[x := 4]
 [y := 1]
 [y := x * y]
 [x := x - a]]
```

1. читаем список строк из файла,
2. в полученном списке разбиваем все элементы на слова,
3. – выбираем только те элементы, у которых второе слово ':=',
4. – у всех элементов оставляем первое слово,

Функциональное решение

```
(define variables
  (o set
     (map first)
     (filter assignment?)
     (map string->words)
     file->lines))
```

```
[[x := 4]
 [y := 1]
 [y := x * y]
 [x := x - a]]
```

1. читаем список строк из файла,
2. в полученном списке разбиваем все элементы на слова,
3. – выбираем только те элементы, у которых второе слово ':=',
4. – у всех элементов оставляем первое слово,

Функциональное решение

```
(define variables
  (o set
     (map first)
     (filter assignment?)
     (map string->words)
     file->lines))
```

```
[x y y x]
```

1. читаем список строк из файла,
2. в полученном списке разбиваем все элементы на слова,
3. – выбираем только те элементы, у которых второе слово ':=',
4. – у всех элементов оставляем первое слово,

Функциональное решение

```
(define variables
  (o set
     (map first)
     (filter assignment?)
     (map string->words)
     file->lines))
```

```
[x y y x]
```

1. читаем список строк из файла,
2. в полученном списке разбиваем все элементы на слова,
3. – выбираем только те элементы, у которых второе слово ':=',
4. – у всех элементов оставляем первое слово,
5. – удаляем повторяющиеся элементы

Функциональное решение

```
(define variables
  (o set
     (map first)
     (filter assignment?)
     (map string->words)
     file->lines))
```

```
[x y y x]
```

1. читаем список строк из файла,
2. в полученном списке разбиваем все элементы на слова,
3. – выбираем только те элементы, у которых второе слово ':=',
4. – у всех элементов оставляем первое слово,
5. – удаляем повторяющиеся элементы

Функциональное решение

```
(define variables
  (o set
     (map first)
     (filter assignment?)
     (map string->words)
     file->lines))
```

```
[x y]
```

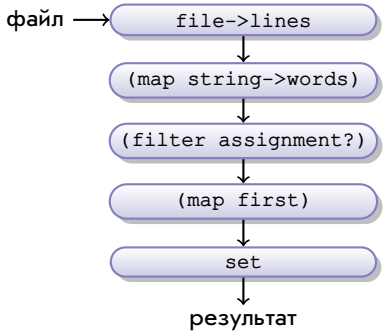
1. читаем список строк из файла,
2. в полученном списке разбиваем все элементы на слова,
3. – выбираем только те элементы, у которых второе слово ':=',
4. – у всех элементов оставляем первое слово,
5. – удаляем повторяющиеся элементы

Функциональное решение

```
(define variables
  (o set
     (map first)
     (filter assignment?)
     (map string->words)
     file->lines))
```

1. читаем список строк из файла,
2. в полученном списке разбиваем все элементы на слова,
3. – выбираем только те элементы, у которых второе слово ':=',
4. – у всех элементов оставляем первое слово,
5. – удаляем повторяющиеся элементы

```
[x y]
```



Абстракция цикла

В функциональном решении нет явных циклов.

Абстракция цикла

В функциональном решении нет явных циклов.

Все комбинируемые операторы работают со списками, как с единым объектом:

Абстракция цикла

В функциональном решении нет явных циклов.

Все комбинируемые операторы работают со списками, как с единым объектом:

$$\texttt{set} :: [A] \to [A]$$
$$(\texttt{map}\ f) :: [A] \to [B]$$
$$(\texttt{filter}\ p) :: [A] \to [A]$$

Абстракция цикла

В функциональном решении нет явных циклов.

Все комбинируемые операторы работают со списками, как с единым объектом:

$$\texttt{set} :: [A] \to [A]$$
$$(\texttt{map}\ f) :: [A] \to [B]$$
$$(\texttt{filter}\ p) :: [A] \to [A]$$

Циклы по элементам списка «спрятаны» от пользователя барьером абстракции.

Вывод рекурсивных уравнений

Оператор отображения

Основное свойство оператора `map`:

$$\texttt{map}\ f\ [a\ b\ c] = [(f\ a)\ (f\ b)\ (f\ c)]$$

Вывод рекурсивных уравнений

Оператор отображения

Основное свойство оператора `map`:

$$\texttt{map}\ f\ [a\ b\ c] = [(f\ a)\ (f\ b)\ (f\ c)]$$

Вывод уравнения:

Вывод рекурсивных уравнений

Оператор отображения

Основное свойство оператора `map`:

$$\texttt{map}\ f\ [a\ b\ c] = [(f\ a)\ (f\ b)\ (f\ c)]$$

Вывод уравнения:

$$\texttt{map}\ f\ a : [b\ c] = (f\ a) : [(f\ b)\ (f\ c)]$$

Вывод рекурсивных уравнений

Оператор отображения

Основное свойство оператора `map`:

$$\texttt{map}\ f\ [a\ b\ c] = [(f\ a)\ (f\ b)\ (f\ c)]$$

Вывод уравнения:

$$\texttt{map}\ f\ a : [b\ c] = (f\ a) : [(f\ b)\ (f\ c)]$$
$$\texttt{map}\ f\ a : [b\ c] = (f\ a) : (\texttt{map}\ f\ [b\ c])$$

Вывод рекурсивных уравнений

Оператор отображения

Основное свойство оператора `map`:

$$\texttt{map}\ f\ [a\ b\ c] = [(f\ a)\ (f\ b)\ (f\ c)]$$

Вывод уравнения:

$$\begin{aligned}
\texttt{map}\ f\ a : [b\ c] &= (f\ a) : [(f\ b)\ (f\ c)] \\
\texttt{map}\ f\ a : [b\ c] &= (f\ a) : (\texttt{map}\ f\ [b\ c]) \\
\texttt{map}\ f\ a : t &= (f\ a) : (\texttt{map}\ f\ t)
\end{aligned}$$

Вывод рекурсивных уравнений

Оператор отображения

Основное свойство оператора `map`:

$$\texttt{map}\ f\ [a\ b\ c] = [(f\ a)\ (f\ b)\ (f\ c)]$$

Вывод уравнения:

$$\begin{aligned}
\texttt{map}\ f\ a : [b\ c] &= (f\ a) : [(f\ b)\ (f\ c)] \\
\texttt{map}\ f\ a : [b\ c] &= (f\ a) : (\texttt{map}\ f\ [b\ c]) \\
\texttt{map}\ f\ a : t &= (f\ a) : (\texttt{map}\ f\ t)
\end{aligned}$$

база рекурсии:

$$\texttt{map}\ f\ [\,] = [\,]$$

Вывод рекурсивных уравнений

Оператор отображения

Основное свойство оператора `map`:

$$\texttt{map}\ f\ [a\ b\ c] = [(f\ a)\ (f\ b)\ (f\ c)]$$

Вывод уравнения:

$$\texttt{map}\ f\ a : [b\ c] = (f\ a) : [(f\ b)\ (f\ c)]$$
$$\texttt{map}\ f\ a : [b\ c] = (f\ a) : (\texttt{map}\ f\ [b\ c])$$
$$\texttt{map}\ f\ a : t = (f\ a) : (\texttt{map}\ f\ t)$$

Определение:

$$\texttt{map}\ f\ a : t = (f\ a) : (\texttt{map}\ f\ t)$$
$$\texttt{map}\ f\ [\,] = [\,]$$

база рекурсии:

$$\texttt{map}\ f\ [\,] = [\,]$$

Вывод рекурсивных уравнений

Функция `set`

Возвращает множество элементов списка.

Вывод рекурсивных уравнений

Функция `set`

Возвращает множество элементов списка.

Как из `set` $[b\ c\ d]$ получить множество `set` $[a\ b\ c\ d]$?

Вывод рекурсивных уравнений

Функция `set`

Возвращает множество элементов списка.

Как из `set` $[b\ c\ d]$ получить множество `set` $[a\ b\ c\ d]$?

$$\texttt{set}\ [a\ b\ c\ d] = \begin{cases} a : set\ [b\ c\ d] & \text{если } a \notin \{b, c, d\}, \\ set\ [b\ c\ d] & \text{иначе;} \end{cases}$$

Вывод рекурсивных уравнений

Функция `set`

Возвращает множество элементов списка.

Как из `set` $[b\ c\ d]$ получить множество `set` $[a\ b\ c\ d]$?

$$\texttt{set}\ [a\ b\ c\ d] = \begin{cases} a : set\ [b\ c\ d] & \text{если } a \notin \{b, c, d\}, \\ set\ [b\ c\ d] & \text{иначе;} \end{cases}$$

база рекурсии: `set` $[\,] = [\,]$.

```
set h : t = if (member? h S)
            then S
            else h : S
            where S = set t
set [] = []
```

Вывод рекурсивных уравнений

Функция `set`

Возвращает множество элементов списка.

Как из `set` $[b\ c\ d]$ получить множество `set` $[a\ b\ c\ d]$?

$$\texttt{set}\ [a\ b\ c\ d] = \begin{cases} a : set\ [b\ c\ d] & \text{если } a \notin \{b, c, d\}, \\ set\ [b\ c\ d] & \text{иначе;} \end{cases}$$

база рекурсии: `set` $[\,] = [\,]$.

```
set h : t = if (member? h S)
            then S
            else h : S
            where S = set t
set [] = []
```

Функуция `filter`

Возвращает список элементов, удовлетворяющих заданному условию.

Вывод рекурсивных уравнений

Функция `set`

Возвращает множество элементов списка.

Как из `set` $[b\ c\ d]$ получить множество `set` $[a\ b\ c\ d]$?

$$\texttt{set}\ [a\ b\ c\ d] = \begin{cases} a : set\ [b\ c\ d] & \text{если } a \notin \{b, c, d\}, \\ set\ [b\ c\ d] & \text{иначе;} \end{cases}$$

база рекурсии: `set` $[\,] = [\,]$.

```
set h : t = if (member? h S)
            then S
            else h : S
            where S = set t
set [] = []
```

Функуция `filter`

Возвращает список элементов, удовлетворяющих заданному условию.

Элемент h может оказаться в фильтрованном списке `filter` $p\ h : t$ только если он удовлетворяет условию p:

Вывод рекурсивных уравнений

Функция `set`

Возвращает множество элементов списка.

Как из `set` $[b\ c\ d]$ получить множество `set` $[a\ b\ c\ d]$?

$$\texttt{set}\ [a\ b\ c\ d] = \begin{cases} a : set\ [b\ c\ d] & \text{если } a \notin \{b, c, d\}, \\ set\ [b\ c\ d] & \text{иначе;} \end{cases}$$

база рекурсии: $\texttt{set}\ [\,] = [\,]$.

```
set h : t = if (member? h S)
            then S
            else h : S
            where S = set t
set [] = []
```

Функуция `filter`

Возвращает список элементов, удовлетворяющих заданному условию.

Элемент h может оказаться в фильтрованном списке `filter` $p\ h : t$ только если он удовлетворяет условию p:

$$\texttt{filter}\ p\ h : t = \begin{cases} h : \texttt{filter}\ p\ t & \text{если } (p\ h), \\ \texttt{filter}\ p\ t & \text{иначе;} \end{cases}$$

Вывод рекурсивных уравнений

Функция `set`

Возвращает множество элементов списка.

Как из `set` $[b\ c\ d]$ получить множество `set` $[a\ b\ c\ d]$?

$$\texttt{set}\ [a\ b\ c\ d] = \begin{cases} a : set\ [b\ c\ d] & \text{если } a \notin \{b, c, d\}, \\ set\ [b\ c\ d] & \text{иначе;} \end{cases}$$

база рекурсии: $\texttt{set}\ [\,] = [\,]$.

```
set h : t = if (member? h S)
            then S
            else h : S
            where S = set t
set [] = []
```

Функция `filter`

Возвращает список элементов, удовлетворяющих заданному условию.

Элемент h может оказаться в фильтрованном списке `filter` $p\ h : t$ только если он удовлетворяет условию p:

$$\texttt{filter}\ p\ h : t = \begin{cases} h : \texttt{filter}\ p\ t & \text{если } (p\ h), \\ \texttt{filter}\ p\ t & \text{иначе;} \end{cases}$$

база рекурсии: $\texttt{filter}\ p\ [\,] = [\,]$.

Вывод рекурсивных уравнений

Функция set

Возвращает множество элементов списка.

Как из `set` $[b\ c\ d]$ получить множество `set` $[a\ b\ c\ d]$?

$$\texttt{set}\ [a\ b\ c\ d] = \begin{cases} a : set\ [b\ c\ d] & \text{если } a \notin \{b, c, d\}, \\ set\ [b\ c\ d] & \text{иначе;} \end{cases}$$

база рекурсии: $\texttt{set}\ [\,] = [\,]$.

```
set h : t = if (member? h S)
            then S
            else h : S
            where S = set t
set [] = []
```

Функция filter

Возвращает список элементов, удовлетворяющих заданному условию.

Элемент h может оказаться в фильтрованном списке `filter` $p\ h : t$ только если он удовлетворяет условию p:

$$\texttt{filter}\ p\ h : t = \begin{cases} h : \texttt{filter}\ p\ t & \text{если } (p\ h), \\ \texttt{filter}\ p\ t & \text{иначе;} \end{cases}$$

база рекурсии: $\texttt{filter}\ p\ [\,] = [\,]$.

```
filter p h : t = if (p h)
                 then h : (filter p t)
                 else (filter p t)
filter p [] = []
```

1 Обработка списков
- Мотивационный пример
- Вывод рекурсивных уравнений

2 Функция свёртки
- Абстракция цикла по списку
- Полезные операторы

3 Обобщение свёртки
- Катаморфизм
- Свёртка для потоков
- Свёртка деревьев

Абстракция цикла по списку

- `set` — последовательно перебирает элементы списка, добавляя или не добавляя элемент в результат, в зависимости от того, содержится ли элемент в списке-результате.

Абстракция цикла по списку

- `set` — последовательно перебирает элементы списка, добавляя или не добавляя элемент в результат, в зависимости от того, содержится ли элемент в списке-результате.
- `(map` f`)` — последовательно перебирает элементы списка, применяя к элементам функцию f и добавляя в результат.

Абстракция цикла по списку

- `set` — последовательно перебирает элементы списка, добавляя или не добавляя элемент в результат, в зависимости от того, содержится ли элемент в списке-результате.
- `(map` f`)` — последовательно перебирает элементы списка, применяя к элементам функцию f и добавляя в результат.
- `(filter` p`)` — последовательно перебирает элементы списка, добавляя или не добавляя элемент в результат, в зависимости от того, удовлетворяет ли элемент условию p.

Абстракция цикла по списку

- `set` — последовательно перебирает элементы списка, добавляя или не добавляя элемент в результат, в зависимости от того, содержится ли элемент в списке-результате.
- `(map` f`)` — последовательно перебирает элементы списка, применяя к элементам функцию f и добавляя в результат.
- `(filter` p`)` — последовательно перебирает элементы списка, добавляя или не добавляя элемент в результат, в зависимости от того, удовлетворяет ли элемент условию p.

Общая схема

Последовательно перебираем элементы списка, комбинируя элемент x и результат res с помощью некоторой функции:

Абстракция цикла по списку

- `set` — последовательно перебирает элементы списка, добавляя или не добавляя элемент в результат, в зависимости от того, содержится ли элемент в списке-результате.
- `(map` f`)` — последовательно перебирает элементы списка, применяя к элементам функцию f и добавляя в результат.
- `(filter` p`)` — последовательно перебирает элементы списка, добавляя или не добавляя элемент в результат, в зависимости от того, удовлетворяет ли элемент условию p.

Общая схема

Последовательно перебираем элементы списка, комбинируя элемент x и результат res с помощью некоторой функции:

- `set`:

$$x, res \mapsto \begin{cases} x : res & \text{если } x \notin res, \\ res & \text{иначе;} \end{cases}$$

Абстракция цикла по списку

- `set` — последовательно перебирает элементы списка, добавляя или не добавляя элемент в результат, в зависимости от того, содержится ли элемент в списке-результате.
- `(map` f`)` — последовательно перебирает элементы списка, применяя к элементам функцию f и добавляя в результат.
- `(filter` p`)` — последовательно перебирает элементы списка, добавляя или не добавляя элемент в результат, в зависимости от того, удовлетворяет ли элемент условию p.

Общая схема

Последовательно перебираем элементы списка, комбинируя элемент x и результат res с помощью некоторой функции:

- `set`:

$$x, res \mapsto \begin{cases} x : res & \text{если } x \notin res, \\ res & \text{иначе;} \end{cases}$$

- `(map` f`)`:

$$x, res \mapsto (f\ x) : res$$

Абстракция цикла по списку

- `set` — последовательно перебирает элементы списка, добавляя или не добавляя элемент в результат, в зависимости от того, содержится ли элемент в списке-результате.
- `(map f)` — последовательно перебирает элементы списка, применяя к элементам функцию f и добавляя в результат.
- `(filter p)` — последовательно перебирает элементы списка, добавляя или не добавляя элемент в результат, в зависимости от того, удовлетворяет ли элемент условию p.

Общая схема

Последовательно перебираем элементы списка, комбинируя элемент x и результат res с помощью некоторой функции:

- `set`:

$$x, res \mapsto \begin{cases} x : res & \text{если } x \notin res, \\ res & \text{иначе;} \end{cases}$$

- `(map f)`:

$$x, res \mapsto (f\ x) : res$$

- `(filter p)`:

$$x, res \mapsto \begin{cases} x : res & \text{если } (p\ x), \\ res & \text{иначе.} \end{cases}$$

Оператор свёртки

Свёртка

Основное свойство оператора свёртки `foldr`:

$$\texttt{foldr} :: (A \times B \to B) \times B \times [A] \to B$$

$$\texttt{foldr}\ f\ x_0\ [a\ b\ c] = f\ a\ (f\ b\ (f\ c\ x_0))$$

Оператор свёртки

Свёртка

Основное свойство оператора свёртки `foldr`:

$$\texttt{foldr} :: (A \times B \to B) \times B \times [A] \to B$$
$$\texttt{foldr}\ f\ x_0\ [a\ b\ c] = f\ a\ (f\ b\ (f\ c\ x_0))$$

Вывод рекурсивных уравнений:

Оператор свёртки

Свёртка

Основное свойство оператора свёртки `foldr`:

$$\texttt{foldr} :: (A \times B \to B) \times B \times [A] \to B$$
$$\texttt{foldr}\ f\ x_0\ [a\ b\ c] = f\ a\ (f\ b\ (f\ c\ x_0))$$

Вывод рекурсивных уравнений:

$$\texttt{foldr}\ f\ x_0\ [a\ b\ c] = f\ a\ (f\ b\ (f\ c\ x_0))$$

Оператор свёртки

Свёртка

Основное свойство оператора свёртки `foldr`:

$$\texttt{foldr} :: (A \times B \to B) \times B \times [A] \to B$$
$$\texttt{foldr}\ f\ x_0\ [a\ b\ c] = f\ a\ (f\ b\ (f\ c\ x_0))$$

Вывод рекурсивных уравнений:

$$\texttt{foldr}\ f\ x_0\ [a\ b\ c] = f\ a\ (f\ b\ (f\ c\ x_0))$$
$$\texttt{foldr}\ f\ x_0\ a : [b\ c] = f\ a\ (\texttt{foldr}\ f\ x_0\ [b\ c])$$

Оператор свёртки

Свёртка

Основное свойство оператора свёртки `foldr`:

$$\texttt{foldr} :: (A \times B \to B) \times B \times [A] \to B$$
$$\texttt{foldr}\ f\ x_0\ [a\ b\ c] = f\ a\ (f\ b\ (f\ c\ x_0))$$

Вывод рекурсивных уравнений:

$$\texttt{foldr}\ f\ x_0\ [a\ b\ c] = f\ a\ (f\ b\ (f\ c\ x_0))$$
$$\texttt{foldr}\ f\ x_0\ a : [b\ c] = f\ a\ (\texttt{foldr}\ f\ x_0\ [b\ c])$$
$$\texttt{foldr}\ f\ x_0\ a : t = f\ a\ (\texttt{foldr}\ f\ x_0\ t)$$

Оператор свёртки

Свёртка

Основное свойство оператора свёртки `foldr`:

$$\texttt{foldr} :: (A \times B \to B) \times B \times [A] \to B$$
$$\texttt{foldr}\ f\ x_0\ [a\ b\ c] = f\ a\ (f\ b\ (f\ c\ x_0))$$

Вывод рекурсивных уравнений:

$$\texttt{foldr}\ f\ x_0\ [a\ b\ c] = f\ a\ (f\ b\ (f\ c\ x_0))$$
$$\texttt{foldr}\ f\ x_0\ a : [b\ c] = f\ a\ (\texttt{foldr}\ f\ x_0\ [b\ c])$$
$$\texttt{foldr}\ f\ x_0\ a : t = f\ a\ (\texttt{foldr}\ f\ x_0\ t)$$

база рекурсии

$$\texttt{foldr}\ f\ x_0\ [\,] = [\,]$$

Оператор свёртки

Свёртка

Основное свойство оператора свёртки `foldr`:

$$\texttt{foldr} :: (A \times B \to B) \times B \times [A] \to B$$
$$\texttt{foldr}\ f\ x_0\ [a\ b\ c] = f\ a\ (f\ b\ (f\ c\ x_0))$$

Определение

$$\texttt{foldr}\ f\ x_0 = F\ \texttt{where}$$
$$F\ [\,] = x_0$$
$$F\ h : t = f\ h\ (F\ t)$$

Вывод рекурсивных уравнений:

$$\texttt{foldr}\ f\ x_0\ [a\ b\ c] = f\ a\ (f\ b\ (f\ c\ x_0))$$
$$\texttt{foldr}\ f\ x_0\ a : [b\ c] = f\ a\ (\texttt{foldr}\ f\ x_0\ [b\ c])$$
$$\texttt{foldr}\ f\ x_0\ a : t = f\ a\ (\texttt{foldr}\ f\ x_0\ t)$$

база рекурсии

$$\texttt{foldr}\ f\ x_0\ [\,] = [\,]$$

Оператор свёртки

Свёртка

Основное свойство оператора свёртки `foldr`:

$$\texttt{foldr} :: (A \times B \to B) \times B \times [A] \to B$$
$$\texttt{foldr}\ f\ x_0\ [a\ b\ c] = f\ a\ (f\ b\ (f\ c\ x_0))$$

Вывод рекурсивных уравнений:

$$\texttt{foldr}\ f\ x_0\ [a\ b\ c] = f\ a\ (f\ b\ (f\ c\ x_0))$$
$$\texttt{foldr}\ f\ x_0\ a : [b\ c] = f\ a\ (\texttt{foldr}\ f\ x_0\ [b\ c])$$
$$\texttt{foldr}\ f\ x_0\ a : t = f\ a\ (\texttt{foldr}\ f\ x_0\ t)$$

база рекурсии

$$\texttt{foldr}\ f\ x_0\ [\,] = [\,]$$

Определение

$$\texttt{foldr}\ f\ x_0 = F\ \texttt{where}$$
$$F\ [\,] = x_0$$
$$F\ h : t = f\ h\ (F\ t)$$

На работу этого оператора можно посмотреть с другой стороны:

Оператор свёртки

Свёртка

Основное свойство оператора свёртки `foldr`:

$$\texttt{foldr} :: (A \times B \to B) \times B \times [A] \to B$$
$$\texttt{foldr}\ f\ x_0\ [a\ b\ c] = f\ a\ (f\ b\ (f\ c\ x_0))$$

Вывод рекурсивных уравнений:

$$\texttt{foldr}\ f\ x_0\ [a\ b\ c] = f\ a\ (f\ b\ (f\ c\ x_0))$$
$$\texttt{foldr}\ f\ x_0\ a : [b\ c] = f\ a\ (\texttt{foldr}\ f\ x_0\ [b\ c])$$
$$\texttt{foldr}\ f\ x_0\ a : t = f\ a\ (\texttt{foldr}\ f\ x_0\ t)$$

база рекурсии

$$\texttt{foldr}\ f\ x_0\ [\,] = [\,]$$

Определение

$$\texttt{foldr}\ f\ x_0 = F\ \texttt{where}$$
$$F\ [\,] = x_0$$
$$F\ h : t = f\ h\ (F\ t)$$

На работу этого оператора можно посмотреть с другой стороны:

$$[a\ b\ c] = \texttt{cons}\ a\ (\texttt{cons}\ b\ (\texttt{cons}\ c\ [\,]))$$

Оператор свёртки

Свёртка

Основное свойство оператора свёртки `foldr`:

$$\texttt{foldr} :: (A \times B \to B) \times B \times [A] \to B$$
$$\texttt{foldr}\ f\ x_0\ [a\ b\ c] = f\ a\ (f\ b\ (f\ c\ x_0))$$

Вывод рекурсивных уравнений:

$$\texttt{foldr}\ f\ x_0\ [a\ b\ c] = f\ a\ (f\ b\ (f\ c\ x_0))$$
$$\texttt{foldr}\ f\ x_0\ a : [b\ c] = f\ a\ (\texttt{foldr}\ f\ x_0\ [b\ c])$$
$$\texttt{foldr}\ f\ x_0\ a : t = f\ a\ (\texttt{foldr}\ f\ x_0\ t)$$

база рекурсии

$$\texttt{foldr}\ f\ x_0\ [\,] = [\,]$$

Определение

$$\texttt{foldr}\ f\ x_0 = F\ \texttt{where}$$
$$F\ [\,] = x_0$$
$$F\ h : t = f\ h\ (F\ t)$$

На работу этого оператора можно посмотреть с другой стороны:

$$[a\ b\ c] = \texttt{cons}\ a\ (\texttt{cons}\ b\ (\texttt{cons}\ c\ [\,]))$$
$$\texttt{foldr}\ f\ x_0\ [a\ b\ c] = \quad f \quad a\ (\ \ f \quad b\ (\ \ f \quad c\ x_0))$$

Оператор свёртки

Свёртка

Основное свойство оператора свёртки `foldr`:

$$\texttt{foldr} :: (A \times B \to B) \times B \times [A] \to B$$
$$\texttt{foldr}\ f\ x_0\ [a\ b\ c] = f\ a\ (f\ b\ (f\ c\ x_0))$$

Вывод рекурсивных уравнений:

$$\texttt{foldr}\ f\ x_0\ [a\ b\ c] = f\ a\ (f\ b\ (f\ c\ x_0))$$
$$\texttt{foldr}\ f\ x_0\ a : [b\ c] = f\ a\ (\texttt{foldr}\ f\ x_0\ [b\ c])$$
$$\texttt{foldr}\ f\ x_0\ a : t = f\ a\ (\texttt{foldr}\ f\ x_0\ t)$$

база рекурсии

$$\texttt{foldr}\ f\ x_0\ [\,] = [\,]$$

Определение

$$\texttt{foldr}\ f\ x_0 = F\ \texttt{where}$$
$$F\ [\,] = x_0$$
$$F\ h : t = f\ h\ (F\ t)$$

На работу этого оператора можно посмотреть с другой стороны:

$$[a\ b\ c] = \texttt{cons}\ a\ (\texttt{cons}\ b\ (\texttt{cons}\ c\ [\,]))$$
$$\texttt{foldr}\ f\ x_0\ [a\ b\ c] = \quad f \quad a\ (\ f \quad b\ (\ f \quad c\ x_0))$$
$$\texttt{cons} \to f$$

Оператор свёртки

Свёртка

Основное свойство оператора свёртки `foldr`:

$$\texttt{foldr} :: (A \times B \to B) \times B \times [A] \to B$$
$$\texttt{foldr}\ f\ x_0\ [a\ b\ c] = f\ a\ (f\ b\ (f\ c\ x_0))$$

Вывод рекурсивных уравнений:

$$\texttt{foldr}\ f\ x_0\ [a\ b\ c] = f\ a\ (f\ b\ (f\ c\ x_0))$$
$$\texttt{foldr}\ f\ x_0\ a : [b\ c] = f\ a\ (\texttt{foldr}\ f\ x_0\ [b\ c])$$
$$\texttt{foldr}\ f\ x_0\ a : t = f\ a\ (\texttt{foldr}\ f\ x_0\ t)$$

база рекурсии

$$\texttt{foldr}\ f\ x_0\ [\,] = [\,]$$

Определение

$$\texttt{foldr}\ f\ x_0 = F\ \texttt{where}$$
$$F\ [\,] = x_0$$
$$F\ h : t = f\ h\ (F\ t)$$

На работу этого оператора можно посмотреть с другой стороны:

$$[a\ b\ c] = \texttt{cons}\ a\ (\texttt{cons}\ b\ (\texttt{cons}\ c\ [\,]))$$
$$\texttt{foldr}\ f\ x_0\ [a\ b\ c] = \quad f \quad a\ (\quad f \quad b\ (\quad f \quad c\ x_0))$$
$$\texttt{cons} \to f$$
$$[\,] \to x_0$$

Оператор свёртки

$$\texttt{map}\ f \quad = \quad \texttt{foldr}\ (x, res \mapsto (f\ x) : res)\ [\,]$$

Оператор свёртки

$$\texttt{map}\ f \quad = \quad \texttt{foldr}\ (x, res \mapsto (f\ x) : res)\ [\,]$$

$$\texttt{set} \quad = \quad \texttt{foldr}\ g\ [\,], \qquad g\ x\ res = \begin{cases} x : res & \textbf{если}\ x \notin res, \\ res & \textbf{иначе}; \end{cases}$$

Оператор свёртки

$$\texttt{map}\ f \quad = \quad \texttt{foldr}\ (x, res \mapsto (f\ x) : res)\ [\,]$$

$$\texttt{set} \quad = \quad \texttt{foldr}\ g\ [\,], \qquad g\ x\ res = \begin{cases} x : res & \textbf{если } x \notin res, \\ res & \textbf{иначе}; \end{cases}$$

$$\texttt{filter}\ p \quad = \quad \texttt{foldr}\ g\ [\,], \qquad g\ x\ res = \begin{cases} x : res & \textbf{если } (p\ x), \\ res & \textbf{иначе}. \end{cases}$$

Оператор свёртки

$$\texttt{map}\ f = \texttt{foldr}\ (x, res \mapsto (f\ x) : res)\ [\,]$$

$$\texttt{set} = \texttt{foldr}\ g\ [\,], \qquad g\ x\ res = \begin{cases} x : res & \textbf{если } x \notin res, \\ res & \textbf{иначе}; \end{cases}$$

$$\texttt{filter}\ p = \texttt{foldr}\ g\ [\,], \qquad g\ x\ res = \begin{cases} x : res & \textbf{если } (p\ x), \\ res & \textbf{иначе}. \end{cases}$$

$$\texttt{length} =$$

Оператор свёртки

$$
\begin{aligned}
\texttt{map}\ f &= \texttt{foldr}\ (x, res \mapsto (f\ x) : res)\ [\,] \\
\texttt{set} &= \texttt{foldr}\ g\ [\,], \qquad g\ x\ res = \begin{cases} x : res & \textbf{если } x \notin res, \\ res & \textbf{иначе}; \end{cases} \\
\texttt{filter}\ p &= \texttt{foldr}\ g\ [\,], \qquad g\ x\ res = \begin{cases} x : res & \textbf{если } (p\ x), \\ res & \textbf{иначе}. \end{cases} \\
\texttt{length} &= \texttt{foldr}\ (x, res \mapsto (1 + res))\ 0
\end{aligned}
$$

Оператор свёртки

$$\texttt{map}\ f = \texttt{foldr}\ (x, res \mapsto (f\ x) : res)\ [\,]$$

$$\texttt{set} = \texttt{foldr}\ g\ [\,], \qquad g\ x\ res = \begin{cases} x : res & \textbf{если } x \notin res, \\ res & \textbf{иначе;} \end{cases}$$

$$\texttt{filter}\ p = \texttt{foldr}\ g\ [\,], \qquad g\ x\ res = \begin{cases} x : res & \textbf{если } (p\ x), \\ res & \textbf{иначе.} \end{cases}$$

$$\texttt{length} = \texttt{foldr}\ (x, res \mapsto (1 + res))\ 0$$

$$\texttt{total} =$$

Оператор свёртки

$$\texttt{map}\ f = \texttt{foldr}\ (x, res \mapsto (f\ x) : res)\ [\,]$$

$$\texttt{set} = \texttt{foldr}\ g\ [\,], \qquad g\ x\ res = \begin{cases} x : res & \textbf{если } x \notin res, \\ res & \textbf{иначе}; \end{cases}$$

$$\texttt{filter}\ p = \texttt{foldr}\ g\ [\,], \qquad g\ x\ res = \begin{cases} x : res & \textbf{если } (p\ x), \\ res & \textbf{иначе}. \end{cases}$$

$$\texttt{length} = \texttt{foldr}\ (x, res \mapsto (1 + res))\ 0$$

$$\texttt{total} = \texttt{foldr}\ (+)\ 0$$

Оператор свёртки

$$\texttt{map}\ f = \texttt{foldr}\ (x, res \mapsto (f\ x) : res)\ [\,]$$

$$\texttt{set} = \texttt{foldr}\ g\ [\,], \qquad g\ x\ res = \begin{cases} x : res & \textbf{если } x \notin res, \\ res & \textbf{иначе;} \end{cases}$$

$$\texttt{filter}\ p = \texttt{foldr}\ g\ [\,], \qquad g\ x\ res = \begin{cases} x : res & \textbf{если } (p\ x), \\ res & \textbf{иначе.} \end{cases}$$

$$\texttt{length} = \texttt{foldr}\ (x, res \mapsto (1 + res))\ 0$$

$$\texttt{total} = \texttt{foldr}\ (+)\ 0$$

$$\texttt{append}\ a\ b =$$

Оператор свёртки

$$
\begin{aligned}
\texttt{map}\ f &= \texttt{foldr}\ (x, res \mapsto (f\ x) : res)\ [\,] \\
\texttt{set} &= \texttt{foldr}\ g\ [\,], \qquad g\ x\ res = \begin{cases} x : res & \textbf{если } x \notin res, \\ res & \textbf{иначе;} \end{cases} \\
\texttt{filter}\ p &= \texttt{foldr}\ g\ [\,], \qquad g\ x\ res = \begin{cases} x : res & \textbf{если } (p\ x), \\ res & \textbf{иначе.} \end{cases} \\
\texttt{length} &= \texttt{foldr}\ (x, res \mapsto (1 + res))\ 0 \\
\texttt{total} &= \texttt{foldr}\ (+)\ 0 \\
\texttt{append}\ a\ b &= \texttt{foldr}\ (:)\ b\ a
\end{aligned}
$$

Оператор свёртки

$$
\begin{aligned}
\texttt{map}\ f &= \texttt{foldr}\ (x, res \mapsto (f\ x) : res)\ [\,] \\
\texttt{set} &= \texttt{foldr}\ g\ [\,], \qquad g\ x\ res = \begin{cases} x : res & \textbf{если } x \notin res, \\ res & \textbf{иначе}; \end{cases} \\
\texttt{filter}\ p &= \texttt{foldr}\ g\ [\,], \qquad g\ x\ res = \begin{cases} x : res & \textbf{если } (p\ x), \\ res & \textbf{иначе}. \end{cases} \\
\texttt{length} &= \texttt{foldr}\ (x, res \mapsto (1 + res))\ 0 \\
\texttt{total} &= \texttt{foldr}\ (+)\ 0 \\
\texttt{append}\ a\ b &= \texttt{foldr}\ (:)\ b\ a \\
\texttt{any?}\ p &=
\end{aligned}
$$

Оператор свёртки

$$
\begin{aligned}
\texttt{map}\ f &= \texttt{foldr}\ (x, res \mapsto (f\ x) : res)\ [\,] \\
\texttt{set} &= \texttt{foldr}\ g\ [\,], \qquad g\ x\ res = \begin{cases} x : res & \textbf{если } x \notin res, \\ res & \textbf{иначе}; \end{cases} \\
\texttt{filter}\ p &= \texttt{foldr}\ g\ [\,], \qquad g\ x\ res = \begin{cases} x : res & \textbf{если } (p\ x), \\ res & \textbf{иначе}. \end{cases} \\
\texttt{length} &= \texttt{foldr}\ (x, res \mapsto (1 + res))\ 0 \\
\texttt{total} &= \texttt{foldr}\ (+)\ 0 \\
\texttt{append}\ a\ b &= \texttt{foldr}\ (:)\ b\ a \\
\texttt{any?}\ p &= \texttt{foldr}\ (x, res \mapsto or\ res\ (p\ x))\ \#f
\end{aligned}
$$

Оператор свёртки

$$
\begin{aligned}
\texttt{map}\ f &= \texttt{foldr}\ (x, res \mapsto (f\ x) : res)\ [\,] \\
\texttt{set} &= \texttt{foldr}\ g\ [\,], \qquad g\ x\ res = \begin{cases} x : res & \textbf{если } x \notin res, \\ res & \textbf{иначе}; \end{cases} \\
\texttt{filter}\ p &= \texttt{foldr}\ g\ [\,], \qquad g\ x\ res = \begin{cases} x : res & \textbf{если } (p\ x), \\ res & \textbf{иначе}. \end{cases} \\
\texttt{length} &= \texttt{foldr}\ (x, res \mapsto (1 + res))\ 0 \\
\texttt{total} &= \texttt{foldr}\ (+)\ 0 \\
\texttt{append}\ a\ b &= \texttt{foldr}\ (:)\ b\ a \\
\texttt{any?}\ p &= \texttt{foldr}\ (x, res \mapsto or\ res\ (p\ x))\ \#f \\
\texttt{member?}\ el &=
\end{aligned}
$$

Оператор свёртки

$$
\begin{aligned}
\texttt{map}\ f &= \texttt{foldr}\ (x, res \mapsto (f\ x) : res)\ [\,] \\
\texttt{set} &= \texttt{foldr}\ g\ [\,], \qquad g\ x\ res = \begin{cases} x : res & \textbf{если } x \notin res, \\ res & \textbf{иначе}; \end{cases} \\
\texttt{filter}\ p &= \texttt{foldr}\ g\ [\,], \qquad g\ x\ res = \begin{cases} x : res & \textbf{если } (p\ x), \\ res & \textbf{иначе}. \end{cases} \\
\texttt{length} &= \texttt{foldr}\ (x, res \mapsto (1 + res))\ 0 \\
\texttt{total} &= \texttt{foldr}\ (+)\ 0 \\
\texttt{append}\ a\ b &= \texttt{foldr}\ (:)\ b\ a \\
\texttt{any?}\ p &= \texttt{foldr}\ (x, res \mapsto or\ res\ (p\ x))\ \#f \\
\texttt{member?}\ el &= \texttt{foldr}\ (x, res \mapsto or\ res\ (x = el))\ \#f
\end{aligned}
$$

Зоопарк полезных операторов

Операторы композиции

$\circ :: (B \to C) \times (A \to B) \to (A \to C)$

$(f \circ g)\ x = f\ (g\ x)$

$f\ \circ\ g = x \mapsto f\ (g\ x)$

Зоопарк полезных операторов

Операторы композиции

$\circ :: (B \to C) \times (A \to B) \to (A \to C)$

$(f \circ g)\ x = f\ (g\ x)$

$f\ \circ\ g = x \mapsto f\ (g\ x)$

$\circ :: (B \times C... \to D) \times (A... \to B) \to (A... \times C... \to D)$

$(f\ \circ\ g)\ x\ y = f\ (g\ x)\ y$

Зоопарк полезных операторов

Операторы композиции

$\circ :: (B \to C) \times (A \to B) \to (A \to C)$

$(f \circ g)\ x = f\ (g\ x)$
$f\ \circ\ g = x \mapsto f\ (g\ x)$

$\circ :: (B \times C... \to D) \times (A... \to B) \to (A... \times C... \to D)$

$(f\ \circ\ g)\ x\ y = f\ (g\ x)\ y$

Оператор каррирования

$\texttt{curry} :: (A \times B \to C) \to (A \to (B \to C))$

$(\texttt{curry}\ f)\ x\ y = x \mapsto (y \mapsto (f\ x\ y))$
$(\texttt{curryr}\ f)\ x\ y = y \mapsto (x \mapsto (f\ x\ y))$

Зоопарк полезных операторов

Операторы композиции

$\circ :: (B \to C) \times (A \to B) \to (A \to C)$

$(f \circ g)\ x = f\ (g\ x)$
$f\ \circ\ g = x \mapsto f\ (g\ x)$

$\circ :: (B \times C... \to D) \times (A... \to B) \to (A... \times C... \to D)$

$(f\ \circ\ g)\ x\ y = f\ (g\ x)\ y$

Оператор каррирования

$\texttt{curry} :: (A \times B \to C) \to (A \to (B \to C))$

$(\texttt{curry}\ f)\ x\ y = x \mapsto (y \mapsto (f\ x\ y))$
$(\texttt{curryr}\ f)\ x\ y = y \mapsto (x \mapsto (f\ x\ y))$

Тривиальная функция

$\texttt{c} :: A \to (B \to A)$

$(\texttt{c}\ a)\ x = a$

Зоопарк полезных операторов

Операторы композиции

$\circ :: (B \to C) \times (A \to B) \to (A \to C)$

$(f \circ g)\ x = f\ (g\ x)$
$f\ \circ\ g = x \mapsto f\ (g\ x)$

$\circ :: (B \times C... \to D) \times (A... \to B) \to (A... \times C... \to D)$

$(f\ \circ\ g)\ x\ y = f\ (g\ x)\ y$

Оператор каррирования

$\texttt{curry} :: (A \times B \to C) \to (A \to (B \to C))$

$(\texttt{curry}\ f)\ x\ y = x \mapsto (y \mapsto (f\ x\ y))$
$(\texttt{curryr}\ f)\ x\ y = y \mapsto (x \mapsto (f\ x\ y))$

Тривиальная функция

$\texttt{c} :: A \to (B \to A)$

$(\texttt{c}\ a)\ x = a$

Оператор перестановки

$\texttt{flip} :: (A \times B \to C) \times (B \times A \to C)$

$(\texttt{flip}\ f)\ x\ y = f\ y\ x$

Зоопарк полезных операторов

Операторы композиции

$\circ :: (B \to C) \times (A \to B) \to (A \to C)$

$(f \circ g)\ x = f\ (g\ x)$
$f\ \circ\ g = x \mapsto f\ (g\ x)$

$\circ :: (B \times C... \to D) \times (A... \to B) \to (A... \times C... \to D)$

$(f\ \circ\ g)\ x\ y = f\ (g\ x)\ y$

Оператор каррирования

$\texttt{curry} :: (A \times B \to C) \to (A \to (B \to C))$

$(\texttt{curry}\ f)\ x\ y = x \mapsto (y \mapsto (f\ x\ y))$
$(\texttt{curryr}\ f)\ x\ y = y \mapsto (x \mapsto (f\ x\ y))$

Тривиальная функция

$\texttt{c} :: A \to (B \to A)$

$(\texttt{c}\ a)\ x = a$

Оператор перестановки

$\texttt{flip} :: (A \times B \to C) \times (B \times A \to C)$

$(\texttt{flip}\ f)\ x\ y = f\ y\ x$

Оператор выбора аргумента

$I_n :: Num \times (A_1 \times A_2 \times ... \to A_n)$

$I_n\ x_1\ x_2\ ... = x_n$

Зоопарк полезных операторов

Оператор отрицания

`negate` $:: (A \to Bool) \to (A \to Bool)$

`negate` $p\ x =$ `not` $(p\ x)$

`negate` $=$ `not` $\circ\ p$

Зоопарк полезных операторов

Оператор отрицания

`negate` :: $(A \to Bool) \to (A \to Bool)$

`negate` $p\ x =$ `not` $(p\ x)$

`negate` $=$ `not` $\circ\ p$

Оператор выбора

`iff` :: $(A \to Bool) \times (A \to B) \times (A \to C) \to (A \to B|C)$

(`iff` $p\ f\ g)\ x =$ **if** $(p\ x)$ **then** $(f\ x)$ **else** $(g\ x)$

Зоопарк полезных операторов

Оператор отрицания

`negate` :: $(A \to Bool) \to (A \to Bool)$

`negate` $p\ x =$ `not` $(p\ x)$

`negate` $=$ `not` $\circ\ p$

Оператор выбора

`iff` :: $(A \to Bool) \times (A \to B) \times (A \to C) \to (A \to B|C)$

(`iff` $p\ f\ g$) $x =$ **if** $(p\ x)$ **then** $(f\ x)$ **else** $(g\ x)$

Операторы конъюнкции и дизъюнкции

`andf, orf` :: $(A \to Bool) \times (A \to Bool) \to (A \to Bool)$

Зоопарк полезных операторов

Оператор отрицания

`negate` :: $(A \to Bool) \to (A \to Bool)$

`negate` $p\ x =$ `not` $(p\ x)$

`negate` $=$ `not` $\circ\ p$

Оператор выбора

`iff` :: $(A \to Bool) \times (A \to B) \times (A \to C) \to (A \to B|C)$

(`iff` $p\ f\ g)\ x =$ **if** $(p\ x)$ **then** $(f\ x)$ **else** $(g\ x)$

Операторы конъюнкции и дизъюнкции

`andf, orf` :: $(A \to Bool) \times (A \to Bool) \to (A \to Bool)$

`andf` $p\ q =$ `iff` $p\ q$ (`C false`)

Зоопарк полезных операторов

Оператор отрицания

`negate` $:: (A \to Bool) \to (A \to Bool)$

`negate` $p\ x =$ `not` $(p\ x)$

`negate` $=$ `not` $\circ\ p$

Оператор выбора

`iff` $:: (A \to Bool) \times (A \to B) \times (A \to C) \to (A \to B|C)$

(`iff` $p\ f\ g)\ x =$ **if** $(p\ x)$ **then** $(f\ x)$ **else** $(g\ x)$

Операторы конъюнкции и дизъюнкции

`andf, orf` $:: (A \to Bool) \times (A \to Bool) \to (A \to Bool)$

`andf` $p\ q =$ `iff` $p\ q$ (`C false`)

`orf` $p\ q =$ `iff` p (`C true`) q

Примеры использования операторов

- Вычисление суммы квадратов чётных чисел от 0 до 100:

Примеры использования операторов

- Вычисление суммы квадратов чётных чисел от 0 до 100:

$$\texttt{sumf}\ (\texttt{sqr} \circ (2\ *))\ 50$$

Примеры использования операторов

- Вычисление суммы квадратов чётных чисел от 0 до 100:

$$\texttt{sumf}\ (\texttt{sqr} \circ (2\ *))\ 50$$

$$\texttt{sumf}\ (\texttt{iff}\ \texttt{odd?}\ sqr\ (\texttt{C}\ 0))\ 100$$

Примеры использования операторов

- Вычисление суммы квадратов чётных чисел от 0 до 100:

$$\texttt{sumf}\ (\texttt{sqr} \circ (2\ *))\ 50$$

$$\texttt{sumf}\ (\texttt{iff odd?}\ sqr\ (\texttt{C}\ 0))\ 100$$

- Бесточечная нотация при нехвостовом вызове:

Примеры использования операторов

- Вычисление суммы квадратов чётных чисел от 0 до 100:

$$\texttt{sumf}\ (\texttt{sqr} \circ (2\ *))\ 50$$
$$\texttt{sumf}\ (\texttt{iff odd?}\ sqr\ (\texttt{C}\ 0))\ 100$$

- Бесточечная нотация при нехвостовом вызове:

$$\texttt{F}\ x\ y = \texttt{G}\ (\texttt{H}\ x\ y)\ a =$$

Примеры использования операторов

- Вычисление суммы квадратов чётных чисел от 0 до 100:

$$\texttt{sumf}\ (\texttt{sqr} \circ (2\ *))\ 50$$
$$\texttt{sumf}\ (\texttt{iff odd?}\ sqr\ (\texttt{C}\ 0))\ 100$$

- Бесточечная нотация при нехвостовом вызове:

$$\texttt{F}\ x\ y = \texttt{G}\ (\texttt{H}\ x\ y)\ a =$$
$$= (\texttt{flip G})\ a\ (\texttt{H}\ x\ y) =$$

Примеры использования операторов

- Вычисление суммы квадратов чётных чисел от 0 до 100:

$$\texttt{sumf}\ (\texttt{sqr} \circ (2\ *))\ 50$$
$$\texttt{sumf}\ (\texttt{iff odd?}\ sqr\ (\texttt{C}\ 0))\ 100$$

- Бесточечная нотация при нехвостовом вызове:

$$\begin{aligned}\texttt{F}\ x\ y &= \texttt{G}\ (\texttt{H}\ x\ y)\ a = \\ &= (\texttt{flip G})\ a\ (\texttt{H}\ x\ y) = \\ &= ((\texttt{flip G})\ a)\ (\texttt{H}\ x\ y) =\end{aligned}$$

Примеры использования операторов

- Вычисление суммы квадратов чётных чисел от 0 до 100:

$$\texttt{sumf}\ (\texttt{sqr} \circ (2\ *))\ 50$$
$$\texttt{sumf}\ (\texttt{iff odd?}\ sqr\ (\texttt{C}\ 0))\ 100$$

- Бесточечная нотация при нехвостовом вызове:

$$\begin{aligned}\texttt{F}\ x\ y &= \texttt{G}\ (\texttt{H}\ x\ y)\ a = \\ &= (\texttt{flip G})\ a\ (\texttt{H}\ x\ y) = \\ &= ((\texttt{flip G})\ a)\ (\texttt{H}\ x\ y) = \\ &= (((\texttt{flip G})\ a) \circ \texttt{H})\ x\ y\end{aligned}$$

Примеры использования операторов

- Вычисление суммы квадратов чётных чисел от 0 до 100:

$$\texttt{sumf}\ (\texttt{sqr} \circ (2\ *))\ 50$$
$$\texttt{sumf}\ (\texttt{iff odd?}\ sqr\ (\texttt{C}\ 0))\ 100$$

- Бесточечная нотация при нехвостовом вызове:

$$\begin{aligned}
\texttt{F}\ x\ y &= \texttt{G}\ (\texttt{H}\ x\ y)\ a = \\
&= (\texttt{flip G})\ a\ (\texttt{H}\ x\ y) = \\
&= ((\texttt{flip G})\ a)\ (\texttt{H}\ x\ y) = \\
&= (((\texttt{flip G})\ a) \circ \texttt{H})\ x\ y \\
\texttt{F} &= ((\texttt{flip G})\ a) \circ \texttt{H}
\end{aligned}$$

Примеры использования операторов

- Вычисление суммы квадратов чётных чисел от 0 до 100:

$$\texttt{sumf}\ (\texttt{sqr} \circ (2\ *))\ 50$$
$$\texttt{sumf}\ (\texttt{iff odd?}\ sqr\ (\texttt{C}\ 0))\ 100$$

- Бесточечная нотация при нехвостовом вызове:

$$\begin{aligned}\texttt{F}\ x\ y &= \texttt{G}\ (\texttt{H}\ x\ y)\ a = \\ &= (\texttt{flip G})\ a\ (\texttt{H}\ x\ y) = \\ &= ((\texttt{flip G})\ a)\ (\texttt{H}\ x\ y) = \\ &= (((\texttt{flip G})\ a) \circ \texttt{H})\ x\ y \\ \texttt{F} &= ((\texttt{flip G})\ a) \circ \texttt{H}\end{aligned}$$

- Комбинаторная запись свёртки:

$$\texttt{map}\ f = \texttt{foldr}\ (\texttt{cons} \circ f)\ []$$

Примеры использования операторов

- Вычисление суммы квадратов чётных чисел от 0 до 100:

$$\texttt{sumf}\ (\texttt{sqr} \circ (2\ *))\ 50$$
$$\texttt{sumf}\ (\texttt{iff odd?}\ sqr\ (\texttt{C}\ 0))\ 100$$

- Бесточечная нотация при нехвостовом вызове:

$$\begin{aligned}\texttt{F}\ x\ y &= \texttt{G}\ (\texttt{H}\ x\ y)\ a = \\ &= (\texttt{flip G})\ a\ (\texttt{H}\ x\ y) = \\ &= ((\texttt{flip G})\ a)\ (\texttt{H}\ x\ y) = \\ &= (((\texttt{flip G})\ a) \circ \texttt{H})\ x\ y \\ \texttt{F} &= ((\texttt{flip G})\ a) \circ \texttt{H}\end{aligned}$$

- Комбинаторная запись свёртки:

$$\begin{aligned}\texttt{map}\ f &= \texttt{foldr}\ (\texttt{cons} \circ f)\ [\,] \\ \texttt{set} &= \texttt{foldr}\ (\texttt{iff member?}\ I_2\ \texttt{cons})\ [\,]\end{aligned}$$

Примеры использования операторов

- Вычисление суммы квадратов чётных чисел от 0 до 100:

$$\texttt{sumf}\ (\texttt{sqr} \circ (2\ *))\ 50$$
$$\texttt{sumf}\ (\texttt{iff odd?}\ sqr\ (\texttt{C}\ 0))\ 100$$

- Бесточечная нотация при нехвостовом вызове:

$$\begin{aligned}\texttt{F}\ x\ y &= \texttt{G}\ (\texttt{H}\ x\ y)\ a = \\ &= (\texttt{flip G})\ a\ (\texttt{H}\ x\ y) = \\ &= ((\texttt{flip G})\ a)\ (\texttt{H}\ x\ y) = \\ &= (((\texttt{flip G})\ a) \circ \texttt{H})\ x\ y \\ \texttt{F} &= ((\texttt{flip G})\ a) \circ \texttt{H}\end{aligned}$$

- Комбинаторная запись свёртки:

$$\begin{aligned}\texttt{map}\ f &= \texttt{foldr}\ (\texttt{cons} \circ f)\ [\,] \\ \texttt{set} &= \texttt{foldr}\ (\texttt{iff member?}\ I_2\ \texttt{cons})\ [\,] \\ \texttt{filter}\ p &= \texttt{foldr}\ (\texttt{iff}\ (p \circ I_1)\ \texttt{cons}\ I_2)\ [\,]\end{aligned}$$

Примеры использования операторов

- Вычисление суммы квадратов чётных чисел от 0 до 100:

$$\texttt{sumf}\ (\texttt{sqr} \circ (2\ *))\ 50$$
$$\texttt{sumf}\ (\texttt{iff odd?}\ sqr\ (\texttt{C}\ 0))\ 100$$

- Бесточечная нотация при нехвостовом вызове:

$$\begin{aligned}\texttt{F}\ x\ y &= \texttt{G}\ (\texttt{H}\ x\ y)\ a = \\ &= (\texttt{flip G})\ a\ (\texttt{H}\ x\ y) = \\ &= ((\texttt{flip G})\ a)\ (\texttt{H}\ x\ y) = \\ &= (((\texttt{flip G})\ a) \circ \texttt{H})\ x\ y \\ \texttt{F} &= ((\texttt{flip G})\ a) \circ \texttt{H}\end{aligned}$$

- Комбинаторная запись свёртки:

$$\begin{aligned}\texttt{map}\ f &= \texttt{foldr}\ (\texttt{cons} \circ f)\ [\,] \\ \texttt{set} &= \texttt{foldr}\ (\texttt{iff member?}\ I_2\ \texttt{cons})\ [\,] \\ \texttt{filter}\ p &= \texttt{foldr}\ (\texttt{iff}\ (p \circ I_1)\ \texttt{cons}\ I_2)\ [\,] \\ \texttt{length} &= \texttt{foldr}\ (+ \circ (\texttt{C}\ 1))\ 0\end{aligned}$$

Примеры использования операторов

- Вычисление суммы квадратов чётных чисел от 0 до 100:

$$\texttt{sumf}\ (\texttt{sqr} \circ (2\ *))\ 50$$
$$\texttt{sumf}\ (\texttt{iff odd?}\ sqr\ (\texttt{C}\ 0))\ 100$$

- Бесточечная нотация при нехвостовом вызове:

$$\begin{aligned}\texttt{F}\ x\ y &= \texttt{G}\ (\texttt{H}\ x\ y)\ a = \\ &= (\texttt{flip G})\ a\ (\texttt{H}\ x\ y) = \\ &= ((\texttt{flip G})\ a)\ (\texttt{H}\ x\ y) = \\ &= (((\texttt{flip G})\ a) \circ \texttt{H})\ x\ y \\ \texttt{F} &= ((\texttt{flip G})\ a) \circ \texttt{H}\end{aligned}$$

- Комбинаторная запись свёртки:

$$\begin{aligned}\texttt{map}\ f &= \texttt{foldr}\ (\texttt{cons} \circ f)\ [] \\ \texttt{set} &= \texttt{foldr}\ (\texttt{iff member?}\ I_2\ \texttt{cons})\ [] \\ \texttt{filter}\ p &= \texttt{foldr}\ (\texttt{iff}\ (p \circ I_1)\ \texttt{cons}\ I_2)\ [] \\ \texttt{length} &= \texttt{foldr}\ (+ \circ (\texttt{C}\ 1))\ 0 \\ \texttt{member?}\ x &= \texttt{foldr}\ (\textbf{or} \circ (= x))\ \texttt{\#f}\end{aligned}$$

Примеры использования операторов

- Вычисление суммы квадратов чётных чисел от 0 до 100:

$$\texttt{sumf}\ (\texttt{sqr} \circ (2\ *))\ 50$$
$$\texttt{sumf}\ (\texttt{iff odd?}\ sqr\ (\texttt{C}\ 0))\ 100$$

- Бесточечная нотация при нехвостовом вызове:

$$\begin{aligned}\texttt{F}\ x\ y &= \texttt{G}\ (\texttt{H}\ x\ y)\ a = \\ &= (\texttt{flip G})\ a\ (\texttt{H}\ x\ y) = \\ &= ((\texttt{flip G})\ a)\ (\texttt{H}\ x\ y) = \\ &= (((\texttt{flip G})\ a) \circ \texttt{H})\ x\ y \\ \texttt{F} &= ((\texttt{flip G})\ a) \circ \texttt{H}\end{aligned}$$

- Комбинаторная запись свёртки:

$$\begin{aligned}\texttt{map}\ f &= \texttt{foldr}\ (\texttt{cons} \circ f)\ [] \\ \texttt{set} &= \texttt{foldr}\ (\texttt{iff member?}\ I_2\ \texttt{cons})\ [] \\ \texttt{filter}\ p &= \texttt{foldr}\ (\texttt{iff}\ (p \circ I_1)\ \texttt{cons}\ I_2)\ [] \\ \texttt{length} &= \texttt{foldr}\ (+ \circ (\texttt{C}\ 1))\ 0 \\ \texttt{member?}\ x &= \texttt{foldr}\ (\textbf{or} \circ (= x))\ \texttt{\#f} \\ \texttt{append} &= \texttt{flip}\ (\texttt{foldr}\ cons)\end{aligned}$$

Левая свёртка списков

Обыкновенная (правая) свёртка абстрагирует рекурсивный процесс обхода списка

$$\texttt{foldr}\ f\ x_0 = F\ \texttt{where}$$
$$F\ [\,] = x_0$$
$$F\ h : t = f\ h\ (F\ t)$$

Левая свёртка списков

Обыкновенная (правая) свёртка абстрагирует рекурсивный процесс обхода списка

$$\begin{aligned}&\texttt{foldr}\ f\ x_0 = F\ \texttt{where}\\ &\qquad F\ [\,] = x_0\\ &\qquad F\ h : t = f\ h\ (F\ t)\end{aligned}$$

Переведём его в итеративный процесс:

$$\begin{aligned}&\texttt{foldl}\ f = F\ \texttt{where}\\ &\qquad F\ x_0\ [\,] = x_0\\ &\qquad F\ res\ h : t = F\ (f\ h\ res)\ t\end{aligned}$$

Левая свёртка списков

Обыкновенная (правая) свёртка абстрагирует рекурсивный процесс обхода списка

$$\begin{aligned}&\texttt{foldr}\ f\ x_0 = F\ \texttt{where}\\ &\quad F\ [\,] = x_0\\ &\quad F\ h : t = f\ h\ (F\ t)\end{aligned}$$

Переведём его в итеративный процесс:

$$\begin{aligned}&\texttt{foldl}\ f = F\ \texttt{where}\\ &\quad F\ x_0\ [\,] = x_0\\ &\quad F\ res\ h : t = F\ (f\ h\ res)\ t\end{aligned}$$

Свойства оператора левой свёртки:

$$\begin{aligned}&\texttt{foldl} :: (A \times B \to B) \times B \times [A] \to B\\ &\texttt{foldl}\ f\ x_0\ [a\ b\ c] = f\ c\ (f\ b\ (f\ a\ x_0))\end{aligned}$$

Левая свёртка списков

Обыкновенная (правая) свёртка абстрагирует рекурсивный процесс обхода списка

$$\begin{aligned}&\texttt{foldr}\ f\ x_0 = F\ \texttt{where}\\ &\quad F\ [\,] = x_0\\ &\quad F\ h : t = f\ h\ (F\ t)\end{aligned}$$

Переведём его в итеративный процесс:

$$\begin{aligned}&\texttt{foldl}\ f = F\ \texttt{where}\\ &\quad F\ x_0\ [\,] = x_0\\ &\quad F\ res\ h : t = F\ (f\ h\ res)\ t\end{aligned}$$

Свойства оператора левой свёртки:

$$\begin{aligned}&\texttt{foldl} :: (A \times B \to B) \times B \times [A] \to B\\ &\texttt{foldl}\ f\ x_0\ [a\ b\ c] = f\ c\ (f\ b\ (f\ a\ x_0))\end{aligned}$$

Примеры использования:

Левая свёртка списков

Обыкновенная (правая) свёртка абстрагирует рекурсивный процесс обхода списка

$$\begin{aligned}&\texttt{foldr}\ f\ x_0 = F\ \texttt{where}\\ &\quad F\ [\,] = x_0\\ &\quad F\ h : t = f\ h\ (F\ t)\end{aligned}$$

Переведём его в итеративный процесс:

$$\begin{aligned}&\texttt{foldl}\ f = F\ \texttt{where}\\ &\quad F\ x_0\ [\,] = x_0\\ &\quad F\ res\ h : t = F\ (f\ h\ res)\ t\end{aligned}$$

Свойства оператора левой свёртки:

$$\begin{aligned}&\texttt{foldl} :: (A \times B \to B) \times B \times [A] \to B\\ &\texttt{foldl}\ f\ x_0\ [a\ b\ c] = f\ c\ (f\ b\ (f\ a\ x_0))\end{aligned}$$

Примеры использования:

$$\texttt{reverse} = \texttt{foldl}\ (:)$$

Левая свёртка списков

Обыкновенная (правая) свёртка абстрагирует рекурсивный процесс обхода списка

$$\begin{aligned}&\texttt{foldr}\ f\ x_0 = F\ \texttt{where}\\ &\quad F\ [\,] = x_0\\ &\quad F\ h : t = f\ h\ (F\ t)\end{aligned}$$

Переведём его в итеративный процесс:

$$\begin{aligned}&\texttt{foldl}\ f = F\ \texttt{where}\\ &\quad F\ x_0\ [\,] = x_0\\ &\quad F\ res\ h : t = F\ (f\ h\ res)\ t\end{aligned}$$

Свойства оператора левой свёртки:

$$\begin{aligned}&\texttt{foldl} :: (A \times B \to B) \times B \times [A] \to B\\ &\texttt{foldl}\ f\ x_0\ [a\ b\ c] = f\ c\ (f\ b\ (f\ a\ x_0))\end{aligned}$$

Примеры использования:

$$\begin{aligned}\texttt{reverse} &= \texttt{foldl}\ (:)\\ \texttt{compose} &= \texttt{foldl}\ (\texttt{flip}\ (\circ))\ id\end{aligned}$$

Левая свёртка списков

Обыкновенная (правая) свёртка абстрагирует рекурсивный процесс обхода списка

$$\begin{aligned}&\texttt{foldr}\ f\ x_0 = F\ \texttt{where}\\ &\quad F\ [\,] = x_0\\ &\quad F\ h : t = f\ h\ (F\ t)\end{aligned}$$

Переведём его в итеративный процесс:

$$\begin{aligned}&\texttt{foldl}\ f = F\ \texttt{where}\\ &\quad F\ x_0\ [\,] = x_0\\ &\quad F\ res\ h : t = F\ (f\ h\ res)\ t\end{aligned}$$

Свойства оператора левой свёртки:

$$\begin{aligned}&\texttt{foldl} :: (A \times B \to B) \times B \times [A] \to B\\ &\texttt{foldl}\ f\ x_0\ [a\ b\ c] = f\ c\ (f\ b\ (f\ a\ x_0))\end{aligned}$$

Примеры использования:

$$\begin{aligned}\texttt{reverse} &= \texttt{foldl}\ (:)\\ \texttt{compose} &= \texttt{foldl}\ (\texttt{flip}\ (\circ))\ id\\ \texttt{pipe} &= \texttt{foldr}\ (\texttt{flip}\ (\circ))\ id\end{aligned}$$

Левая свёртка списков

Обыкновенная (правая) свёртка абстрагирует рекурсивный процесс обхода списка

$$\begin{aligned}&\texttt{foldr}\ f\ x_0 = F\ \texttt{where}\\ &\quad F\ [\,] = x_0\\ &\quad F\ h : t = f\ h\ (F\ t)\end{aligned}$$

Переведём его в итеративный процесс:

$$\begin{aligned}&\texttt{foldl}\ f = F\ \texttt{where}\\ &\quad F\ x_0\ [\,] = x_0\\ &\quad F\ res\ h : t = F\ (f\ h\ res)\ t\end{aligned}$$

Свойства оператора левой свёртки:

$$\begin{aligned}&\texttt{foldl} :: (A \times B \to B) \times B \times [A] \to B\\ &\texttt{foldl}\ f\ x_0\ [a\ b\ c] = f\ c\ (f\ b\ (f\ a\ x_0))\end{aligned}$$

Примеры использования:

$$\begin{aligned}\texttt{reverse} &= \texttt{foldl}\ (:)\\ \texttt{compose} &= \texttt{foldl}\ (\texttt{flip}\ (\circ))\ id\\ \texttt{pipe} &= \texttt{foldr}\ (\texttt{flip}\ (\circ))\ id\end{aligned}$$

$$\begin{aligned}\texttt{compose}\ [f\ g\ h] &= ((id \circ f) \circ g) \circ h = f \circ g \circ h\\ \texttt{pipe}\ [f\ g\ h] &= ((id \circ h) \circ g) \circ f = h \circ g \circ f\end{aligned}$$

Свойства свертки

1. Дуализм левой и правой свёртки:

$$\texttt{foldr}\ f\ x_0\ [a\ b\ c] = f\ a\ (f\ b\ (f\ c\ x_0))$$
$$\texttt{foldl}\ f\ x_0\ [a\ b\ c] = f\ c\ (f\ b\ (f\ a\ x_0))$$

Свойства свертки

1. Дуализм левой и правой свёртки:

$$\texttt{foldr}\ f\ x_0\ [a\ b\ c] = f\ a\ (f\ b\ (f\ c\ x_0))$$
$$\texttt{foldl}\ f\ x_0\ [a\ b\ c] = f\ c\ (f\ b\ (f\ a\ x_0))$$

2. Левая свёртка может быть выражена через правую и наоборот:

$$K\ f\ x\ y = z \mapsto y\ (f\ z\ x)$$
$$\texttt{foldl}\ f\ x_0\ lst = (\texttt{foldr}\ (K\ f)\ \texttt{id}\ lst)\ x_0$$
$$\texttt{foldr}\ f\ x_0\ lst = (\texttt{foldl}\ (K\ f)\ \texttt{id}\ lst)\ x_0$$

Свойства свертки

1. Дуализм левой и правой свёртки:

$$\texttt{foldr}\ f\ x_0\ [a\ b\ c] = f\ a\ (f\ b\ (f\ c\ x_0))$$
$$\texttt{foldl}\ f\ x_0\ [a\ b\ c] = f\ c\ (f\ b\ (f\ a\ x_0))$$

2. Левая свёртка может быть выражена через правую и наоборот:

$$K\ f\ x\ y = z \mapsto y\ (f\ z\ x)$$
$$\texttt{foldl}\ f\ x_0\ lst = (\texttt{foldr}\ (K\ f)\ \texttt{id}\ lst)\ x_0$$
$$\texttt{foldr}\ f\ x_0\ lst = (\texttt{foldl}\ (K\ f)\ \texttt{id}\ lst)\ x_0$$

Обе свёртки, выраженные таким образом, реализуют рекурсивный процесс.

Свойства свертки

1. Дуализм левой и правой свёртки:

$$\texttt{foldr}\ f\ x_0\ [a\ b\ c] = f\ a\ (f\ b\ (f\ c\ x_0))$$
$$\texttt{foldl}\ f\ x_0\ [a\ b\ c] = f\ c\ (f\ b\ (f\ a\ x_0))$$

2. Левая свёртка может быть выражена через правую и наоборот:

$$K\ f\ x\ y = z \mapsto y\ (f\ z\ x)$$
$$\texttt{foldl}\ f\ x_0\ lst = (\texttt{foldr}\ (K\ f)\ \texttt{id}\ lst)\ x_0$$
$$\texttt{foldr}\ f\ x_0\ lst = (\texttt{foldl}\ (K\ f)\ \texttt{id}\ lst)\ x_0$$

Обе свёртки, выраженные таким образом, реализуют рекурсивный процесс.

3. Условия эквивалентности левой и правой свёртки:

$$\texttt{foldr}\ f = \texttt{foldl}\ f,$$

если f коммутативна и ассоциативна:

$$f\ x\ y = f\ y\ x$$
$$f\ x\ (f\ y\ z) = f\ (f\ x\ y)\ z$$

Свойства свертки

1. Дуализм левой и правой свёртки:

$$\texttt{foldr}\ f\ x_0\ [a\ b\ c] = f\ a\ (f\ b\ (f\ c\ x_0))$$
$$\texttt{foldl}\ f\ x_0\ [a\ b\ c] = f\ c\ (f\ b\ (f\ a\ x_0))$$

2. Левая свёртка может быть выражена через правую и наоборот:

$$K\ f\ x\ y = z \mapsto y\ (f\ z\ x)$$
$$\texttt{foldl}\ f\ x_0\ lst = (\texttt{foldr}\ (K\ f)\ \texttt{id}\ lst)\ x_0$$
$$\texttt{foldr}\ f\ x_0\ lst = (\texttt{foldl}\ (K\ f)\ \texttt{id}\ lst)\ x_0$$

Обе свёртки, выраженные таким образом, реализуют рекурсивный процесс.

3. Условия эквивалентности левой и правой свёртки:

$$\texttt{foldr}\ f = \texttt{foldl}\ f,$$

если f коммутативна и ассоциативна:

$$f\ x\ y = f\ y\ x$$
$$f\ x\ (f\ y\ z) = f\ (f\ x\ y)\ z$$

4. Свёртка позволяет организовывать параллельную обработку

$$\texttt{fold}\ f\ x_0\ a :: b = f\ (\texttt{fold}\ f\ x_0\ a)\ (\texttt{fold}\ f\ x_0\ b),$$

если f ассоциативна и x_0 её нейтральный элемент.

1 Обработка списков
- Мотивационный пример
- Вывод рекурсивных уравнений

2 Функция свёртки
- Абстракция цикла по списку
- Полезные операторы

3 Обобщение свёртки
- Катаморфизм
- Свёртка для потоков
- Свёртка деревьев

Катаморфизм, как абстракция структурной рекурсии

Свёртка:

$$\texttt{foldr}\ f\ x_0 = F$$
$$\quad \texttt{where}\ F\ [\,] = x_0$$
$$\qquad\qquad F\ l = f\ (\texttt{head}\ l)\ (F\ (\texttt{tail}\ l))$$

Катаморфизм, как абстракция структурной рекурсии

Свёртка:

$$\begin{aligned}&\texttt{foldr}\ f\ x_0 = F\\ &\quad \texttt{where}\ F\ [\,] = x_0\\ &\qquad\quad F\ l = f\ (\texttt{head}\ l)\ (F\ (\texttt{tail}\ l))\end{aligned}$$

Целочисленный итератор (аккумулятор):

$$\begin{aligned}&\texttt{accum}\ g\ f\ x_0 = F\\ &\quad \texttt{where}\ F\ 0 = x_0\\ &\qquad\quad F\ n = f\ (g\ n)\ (F\ (n-1))\end{aligned}$$

Катаморфизм, как абстракция структурной рекурсии

Свёртка:

$$\begin{aligned}&\texttt{foldr}\ f\ x_0 = F\\ &\quad\texttt{where}\ F\ [\,] = x_0\\ &\qquad\quad F\ l = f\ (\texttt{head}\ l)\ (F\ (\texttt{tail}\ l))\end{aligned}$$

Целочисленный итератор (аккумулятор):

$$\begin{aligned}&\texttt{accum}\ g\ f\ x_0 = F\\ &\quad\texttt{where}\ F\ 0 = x_0\\ &\qquad\quad F\ n = f\ (g\ n)\ (F\ (n-1))\end{aligned}$$

Оба эти оператора реализуют рекурсию по индуктивному типу данных:

Катаморфизм, как абстракция структурной рекурсии

Свёртка:

$$\begin{aligned}&\texttt{foldr}\ f\ x_0 = F\\ &\quad\texttt{where}\ F\ [\,] = x_0\\ &\qquad\quad F\ l = f\ (\texttt{head}\ l)\ (F\ (\texttt{tail}\ l))\end{aligned}$$

Целочисленный итератор (аккумулятор):

$$\begin{aligned}&\texttt{accum}\ g\ f\ x_0 = F\\ &\quad\texttt{where}\ F\ 0 = x_0\\ &\qquad\quad F\ n = f\ (g\ n)\ (F\ (n-1))\end{aligned}$$

Оба эти оператора реализуют рекурсию по индуктивному типу данных:

```
List ::= []
       | A : List
```

$$\begin{aligned}base &= [\,]\\ curr &= \texttt{head}\\ pred &= \texttt{tail}\end{aligned}$$

Катаморфизм, как абстракция структурной рекурсии

Свёртка:

$$\begin{aligned}&\texttt{foldr}\ f\ x_0 = F\\ &\quad\texttt{where}\ F\ [\,] = x_0\\ &\qquad\quad F\ l = f\ (\texttt{head}\ l)\ (F\ (\texttt{tail}\ l))\end{aligned}$$

Целочисленный итератор (аккумулятор):

$$\begin{aligned}&\texttt{accum}\ g\ f\ x_0 = F\\ &\quad\texttt{where}\ F\ 0 = x_0\\ &\qquad\quad F\ n = f\ (g\ n)\ (F\ (n-1))\end{aligned}$$

Оба эти оператора реализуют рекурсию по индуктивному типу данных:

```
List ::= []                 Int ::= 0
       | A : List                 | 1 + Int

base  = []                  base = 0
curr  = head                curr  = id
pred  = tail                pred  = (- 1)
```

Катаморфизм, как абстракция структурной рекурсии

Свёртка:

$$\begin{aligned}&\texttt{foldr}\ f\ x_0 = F\\ &\quad\texttt{where}\ F\ [\,] = x_0\\ &\qquad\quad F\ l = f\ (\texttt{head}\ l)\ (F\ (\texttt{tail}\ l))\end{aligned}$$

Целочисленный итератор (аккумулятор):

$$\begin{aligned}&\texttt{accum}\ g\ f\ x_0 = F\\ &\quad\texttt{where}\ F\ 0 = x_0\\ &\qquad\quad F\ n = f\ (g\ n)\ (F\ (n-1))\end{aligned}$$

Оба эти оператора реализуют рекурсию по индуктивному типу данных:

```
List ::= []                 Int ::= 0
       | A : List                 | 1 + Int

base  = []                  base = 0
curr  = head                curr  = id
pred  = tail                pred  = (- 1)
```

Оператор катаморфизма (обобщённой свёртки):

$$\begin{aligned}&\texttt{cat}\ base\ pred\ curr\ f\ x_0 = F\\ &\quad\texttt{where}\ F\ base = x_0\\ &\qquad\quad F\ x = f\ (curr\ x)\ (F\ (pred\ x))\end{aligned}$$

Катаморфизм, как абстракция структурной рекурсии

Свёртка:

$$\begin{aligned}&\texttt{foldr}\ f\ x_0 = F\\ &\quad\texttt{where}\ F\ [\,] = x_0\\ &\qquad\quad F\ l = f\ (\texttt{head}\ l)\ (F\ (\texttt{tail}\ l))\\ &\texttt{foldr} = \texttt{cat}\ [\,]\ \texttt{tail head}\end{aligned}$$

Целочисленный итератор (аккумулятор):

$$\begin{aligned}&\texttt{accum}\ g\ f\ x_0 = F\\ &\quad\texttt{where}\ F\ 0 = x_0\\ &\qquad\quad F\ n = f\ (g\ n)\ (F\ (n-1))\end{aligned}$$

Оба эти оператора реализуют рекурсию по индуктивному типу данных:

```
List ::= []                 Int ::= 0
       | A : List                 | 1 + Int

base  = []                  base = 0
curr  = head                curr  = id
pred  = tail                pred  = (- 1)
```

Оператор катаморфизма (обобщённой свёртки):

$$\begin{aligned}&\texttt{cat}\ base\ pred\ curr\ f\ x_0 = F\\ &\quad\texttt{where}\ F\ base = x_0\\ &\qquad\quad F\ x = f\ (curr\ x)\ (F\ (pred\ x))\end{aligned}$$

Катаморфизм, как абстракция структурной рекурсии

Свёртка:

$$\begin{aligned}&\texttt{foldr}\ f\ x_0 = F\\ &\quad\texttt{where}\ F\ [\,] = x_0\\ &\qquad\quad F\ l = f\ (\texttt{head}\ l)\ (F\ (\texttt{tail}\ l))\\ &\texttt{foldr} = \texttt{cat}\ [\,]\ \texttt{tail head}\end{aligned}$$

Целочисленный итератор (аккумулятор):

$$\begin{aligned}&\texttt{accum}\ g\ f\ x_0 = F\\ &\quad\texttt{where}\ F\ 0 = x_0\\ &\qquad\quad F\ n = f\ (g\ n)\ (F\ (n-1))\\ &\texttt{accum} = \texttt{cat}\ 0\ (-1)\end{aligned}$$

Оба эти оператора реализуют рекурсию по индуктивному типу данных:

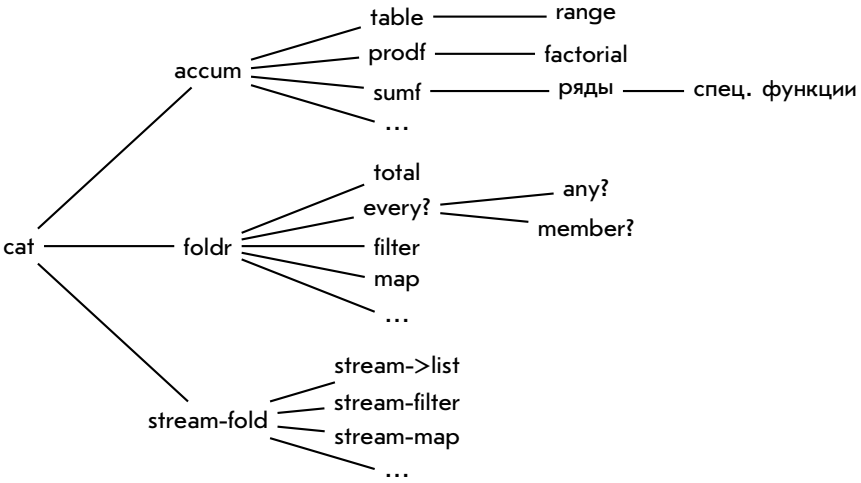
```
List ::= []                 Int ::= 0
       | A : List                 | 1 + Int

base  = []                  base = 0
curr  = head                curr  = id
pred  = tail                pred  = (- 1)
```

Оператор катаморфизма (обобщённой свёртки):

$$\begin{aligned}&\texttt{cat}\ base\ pred\ curr\ f\ x_0 = F\\ &\quad\texttt{where}\ F\ base = x_0\\ &\qquad\quad F\ x = f\ (curr\ x)\ (F\ (pred\ x))\end{aligned}$$

Катаморфизм, как абстракция структурной рекурсии



Использование обобщённой свёртки для потоков

Определение

Поток – последовательнось элементов, вызываемых (создаваемых) по требованию.

Использование обобщённой свёртки для потоков

Определение

Поток – последовательнось элементов, вызываемых (создаваемых) по требованию.

Пример: чтение из порта.

Базовые функции:

```
read-word :: Port → String
read-line :: Port → String
```

Использование обобщённой свёртки для потоков

Определение

Поток – последовательнось элементов, вызываемых (создаваемых) по требованию.

Пример: чтение из порта.

Базовые функции:

```
read-word :: Port → String
read-line :: Port → String
```

Определение индуктивных типов для потоков слов и строк из порта:

```
words p = cons* (read-word p) (words p)
lines p = cons* (read-line p) (lines p)

base  = <eof>
pred  = tail*
curr  = head*
```

Использование обобщённой свёртки для потоков

Определение

Поток – последовательнось элементов, вызываемых (создаваемых) по требованию.

Пример: чтение из порта.

Базовые функции:

```
read-word :: Port → String
read-line :: Port → String
```

Определение индуктивных типов для потоков слов и строк из порта:

```
words p = cons* (read-word p) (words p)
lines p = cons* (read-line p) (lines p)

base  = <eof>
pred  = tail*
curr  = head*
```

Свёртка потока

```
fold* = cat (eof? ∘ head*) tail* head*
```

Использование обобщённой свёртки для потоков

Определение

Поток – последовательнось элементов, вызываемых (создаваемых) по требованию.

Пример: чтение из порта.

Базовые функции:

```
read-word :: Port → String
read-line :: Port → String
```

Определение индуктивных типов для потоков слов и строк из порта:

```
words p = cons* (read-word p) (words p)
lines p = cons* (read-line p) (lines p)

base  = <eof>
pred  = tail*
curr  = head*
```

Свёртка потока

```
fold* = cat (eof? ∘ head*) tail* head*
```

Функции обработки потоков

Отображение и фильтрация:

```
map* f = fold* (cons* ∘ f) <eof>

filter* p =
  fold* (iff (p ∘ I1) cons I2) <eof>

set* p = fold* insert <eof>
```

Использование обобщённой свёртки для потоков

Определение

Поток – последовательнось элементов, вызываемых (создаваемых) по требованию.

Пример: чтение из порта.

Базовые функции:

```
read-word :: Port → String
read-line :: Port → String
```

Определение индуктивных типов для потоков слов и строк из порта:

```
words p = cons* (read-word p) (words p)
lines p = cons* (read-line p) (lines p)

base  = <eof>
pred  = tail*
curr  = head*
```

Свёртка потока

```
fold* = cat (eof? ∘ head*) tail* head*
```

Функции обработки потоков

Отображение и фильтрация:

```
map* f = fold* (cons* ∘ f) <eof>

filter* p =
  fold* (iff (p ∘ I1) cons I2) <eof>

set* p = fold* insert <eof>
```

Преобразование потока в список:

```
stream->list = fold* cons []
```

Обработка файлов

Обработка списков

```
(define variables
  (o set
     (map first)
     (filter assignment?)
     (map string->words)
     file->lines))
```

Обработка файлов

Обработка списков

```
(define variables
  (o set
     (map first)
     (filter assignment?)
     (map string->words)
     file->lines))

(define file->lines
  (o stream->list
     lines
     open-input-file))
```

Обработка файлов

Обработка списков

```
(define variables
  (o set
     (map first)
     (filter assignment?)
     (map string->words)
     file->lines))

(define file->lines
  (o stream->list
     lines
     open-input-file))

(define string->words
  (o stream->list
     words
     open-input-string))
```

Обработка файлов

Обработка списков

```
(define variables
  (o set
     (map first)
     (filter assignment?)
     (map string->words)
     file->lines))

(define file->lines
  (o stream->list
     lines
     open-input-file))

(define string->words
  (o stream->list
     words
     open-input-string))
```

Обработка потоков

```
(define variables
  (o set*
     (map* head*)
     (filter* assignment?)
     (map* string->words)
     file->lines))
```

Обработка файлов

Обработка списков

```
(define variables
  (o set
     (map first)
     (filter assignment?)
     (map string->words)
     file->lines))

(define file->lines
  (o stream->list
     lines
     open-input-file))

(define string->words
  (o stream->list
     words
     open-input-string))
```

Обработка потоков

```
(define variables
  (o set*
     (map* head*)
     (filter* assignment?)
     (map* string->words)
     file->lines))

(define file->lines
  (o lines
     open-input-file))
```

Обработка файлов

Обработка списков

```
(define variables
  (o set
     (map first)
     (filter assignment?)
     (map string->words)
     file->lines))

(define file->lines
  (o stream->list
     lines
     open-input-file))

(define string->words
  (o stream->list
     words
     open-input-string))
```

Обработка потоков

```
(define variables
  (o set*
     (map* head*)
     (filter* assignment?)
     (map* string->words)
     file->lines))

(define file->lines
  (o lines
     open-input-file))

(define string->words
  (o words
     open-input-string))
```

Свёртка деревьев

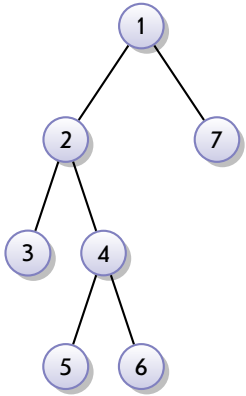
Размеченное двоичное дерево

```
BTree ::= Empty
        | (Node Atom BTree BTree)
```

Свёртка деревьев

Размеченное двоичное дерево

```
BTree ::= Empty
        | (Node Atom BTree BTree)
```



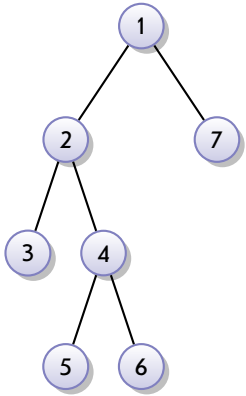
Свёртка деревьев

Размеченное двоичное дерево

```
BTree ::= Empty
        | (Node Atom BTree BTree)
```

Оператор свёртки

```
foldt f g x0 = F
  where F Empty = x0
        F (Node d l r) = f (g d) (F l) (F r)
```



Свёртка деревьев

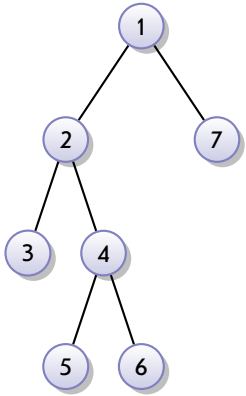
Размеченное двоичное дерево

```
BTree ::= Empty
        | (Node Atom BTree BTree)
```

Оператор свёртки

```
foldt f g x0 = F
  where F Empty = x0
        F (Node d l r) = f (g d) (F l) (F r)

flatten    = foldt append (:[]) []
total      = foldt + id 0
map-tree f = foldt node f []
elements p = foldt append (iff p (:[]) []) []
```



Свёртка деревьев

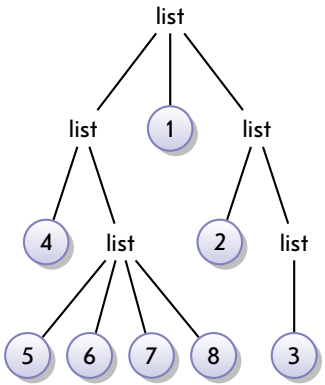
S-выражение

```
SExpr ::= Atom
        | [SExpr]
```

Свёртка деревьев

S-выражение

```
SExpr ::= Atom
        | [SExpr]
```



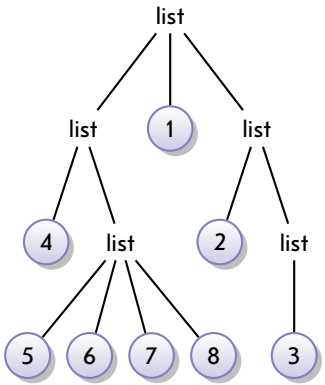
Свёртка деревьев

S-выражение

```
SExpr ::= Atom
        | [SExpr]
```

Оператор свёртки

```
folds f g x0 = F
  where F [t...] = foldr (f ∘ F) x0 t
        F t = g t
```



Свёртка деревьев

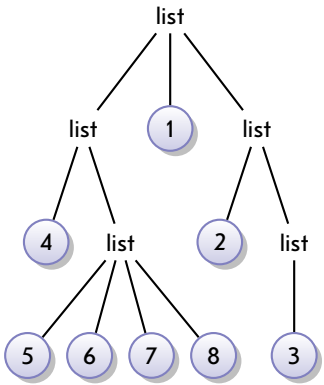
S-выражение

```
SExpr ::= Atom
        | [SExpr]
```

Оператор свёртки

```
folds f g x0 = F
  where F [t...] = foldr (f ∘ F) x0 t
        F t = g t

flatten     = folds append (:[]) []
total       = folds + id 0
map-tree f  = folds : f []
elements p  = foldt append (iff p (:[]) []) []
```



Свёртка деревьев

Размеченное двоичное дерево

```
BTree ::= []
        | (node Atom BTree BTree)
```

S-выражение

```
SExpr ::= Atom
        | [SExpr]
```

Свёртка деревьев

Размеченное двоичное дерево

```
BTree ::= []
        | (node Atom BTree BTree)
```

S-выражение

```
SExpr ::= Atom
        | [SExpr]
```

Свёртка деревьев

Размеченное двоичное дерево

```
BTree ::= []
        | (node Atom BTree BTree)
```

S-выражение

```
SExpr ::= Atom
        | [SExpr]
```

Оператор свёртки

```
foldt f g x0 = F
 where F [] = x0
       F (node d l r) = f (g d) (F l) (F r)
```

Оператор свёртки

```
folds f g x0 = F
  where F [t...] = foldr (f ∘ F) x0 t
        F t = g t
```

Свёртка деревьев

Размеченное двоичное дерево

```
BTree ::= []
        | (node Atom BTree BTree)
```

S-выражение

```
SExpr ::= Atom
        | [SExpr]
```

Оператор свёртки

```
foldt f g x0 = F
 where F [] = x0
       F (node d l r) = f (g d) (F l) (F r)
```

Оператор свёртки

```
folds f g x0 = F
  where F [t...] = foldr (f ∘ F) x0 t
        F t = g t
```

Функции обработки дерева

```
flatten    = foldt append (:[]) []
total      = foldt + id 0
map-tree f = foldt node f []
elements p = foldt append (iff p (:[]) []) []
```

Функции обработки дерева

```
flatten    = folds append (:[]) []
total      = folds + id 0
map-tree f = folds : f []
elements p = foldt append (iff p (:[]) []) []
```

Подведение итогов

Преимущества использования свёртки:

Подведение итогов

Преимущества использования свёртки:

- Гарантированная завершаемость рекурсии.

Подведение итогов

Преимущества использования свёртки:

- Гарантированная завершаемость рекурсии.
- Инкапсуляция деталей реализации и обработки индуктивных типов данных.

Подведение итогов

Преимущества использования свёртки:

- Гарантированная завершаемость рекурсии.
- Инкапсуляция деталей реализации и обработки индуктивных типов данных.
- Высокая степень модульности и выразительности программ.

Подведение итогов

Преимущества использования свёртки:

- Гарантированная завершаемость рекурсии.
- Инкапсуляция деталей реализации и обработки индуктивных типов данных.
- Высокая степень модульности и выразительности программ.
- Стандартизация интерфейсов.

Подведение итогов

Преимущества использования свёртки:

- Гарантированная завершаемость рекурсии.
- Инкапсуляция деталей реализации и обработки индуктивных типов данных.
- Высокая степень модульности и выразительности программ.
- Стандартизация интерфейсов.

Ограничения свёртки:

Подведение итогов

Преимущества использования свёртки:

- Гарантированная завершаемость рекурсии.
- Инкапсуляция деталей реализации и обработки индуктивных типов данных.
- Высокая степень модульности и выразительности программ.
- Стандартизация интерфейсов.

Ограничения свёртки:

- Индуктивная структура обрабатывается полностью, независимо от необходимости.

Подведение итогов

Преимущества использования свёртки:

- Гарантированная завершаемость рекурсии.
- Инкапсуляция деталей реализации и обработки индуктивных типов данных.
- Высокая степень модульности и выразительности программ.
- Стандартизация интерфейсов.

Ограничения свёртки:

- Индуктивная структура обрабатывается полностью, независимо от необходимости.
- База рекурсии может быть только одна.

Подведение итогов

Преимущества использования свёртки:

- Гарантированная завершаемость рекурсии.
- Инкапсуляция деталей реализации и обработки индуктивных типов данных.
- Высокая степень модульности и выразительности программ.
- Стандартизация интерфейсов.

Ограничения свёртки:

- Индуктивная структура обрабатывается полностью, независимо от необходимости.
- База рекурсии может быть только одна.
- Не всякий цикл сводится к свёртке.

Подведение итогов

Преимущества использования свёртки:

- Гарантированная завершаемость рекурсии.
- Инкапсуляция деталей реализации и обработки индуктивных типов данных.
- Высокая степень модульности и выразительности программ.
- Стандартизация интерфейсов.

Ограничения свёртки:

- Индуктивная структура обрабатывается полностью, независимо от необходимости.
- База рекурсии может быть только одна.
- Не всякий цикл сводится к свёртке.

Разработка абстрактного алгебраического типа данных.

АДТ

Подведение итогов

Преимущества использования свёртки:

- Гарантированная завершаемость рекурсии.
- Инкапсуляция деталей реализации и обработки индуктивных типов данных.
- Высокая степень модульности и выразительности программ.
- Стандартизация интерфейсов.

Ограничения свёртки:

- Индуктивная структура обрабатывается полностью, независимо от необходимости.
- База рекурсии может быть только одна.
- Не всякий цикл сводится к свёртке.

Разработка абстрактного алгебраического типа данных.

АДТ $\Longrightarrow$ Определение свёртки

Подведение итогов

Преимущества использования свёртки:

- Гарантированная завершаемость рекурсии.
- Инкапсуляция деталей реализации и обработки индуктивных типов данных.
- Высокая степень модульности и выразительности программ.
- Стандартизация интерфейсов.

Ограничения свёртки:

- Индуктивная структура обрабатывается полностью, независимо от необходимости.
- База рекурсии может быть только одна.
- Не всякий цикл сводится к свёртке.

Разработка абстрактного алгебраического типа данных.

АДТ $\Longrightarrow$ Определение свёртки $\Longrightarrow$ Определение функций обработки АДТ.