

Алгоритмы и структуры данных

Лекция 2 (продолжение)

Раздел:

Рекурсивная обработка списков

Тема Лекции:

Рекурсивная обработка линейных списков

Напоминание материала 1-го курса о линейных списках

Л1-список (линейный однонаправленный)
как абстрактный тип данных (АТД)
определяется через класс базовых
операций, которые (и только они)
выполняются над Л1-списком.

Все остальные действия над Л1-списком
реализуются на основе этих операций.



Об абстракции при разработке программ

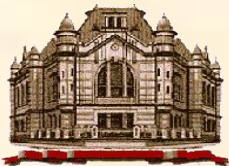
Каждому этапу разработки программы соответствует свой уровень детализации описаний **алгоритмов** и **данных**.

К процедурным абстракциям относятся **абстракция через параметризацию** и **абстракция через спецификацию**.

Они позволяют добавлять новые операции к базовым операциям языка программирования.

Может потребоваться и добавление новых **типов данных**.

Абстракция данных – это совокупность описания данных и набора операций, задающих способ работы с ними.



Абстракция данных

Спецификация абстрактных данных – это:

Заголовок – имя абстрактного типа и перечисление допустимых операций

Комментарий

секция 1 – структура и возможные значения

секция 2 – процедурные спецификации допустимых операций

Тип данных описывается не с помощью языка программирования, а в математических терминах или в терминах других компонентов данных, которые понятны тем, для кого предназначены спецификации



Использование спецификаций в процессе конструирования программ позволяет:

- Формулировать задачу
- Описывать её декомпозицию
- Изменять уровень детализации без учёта реализации (на каждом уровне детализации свои спецификации).

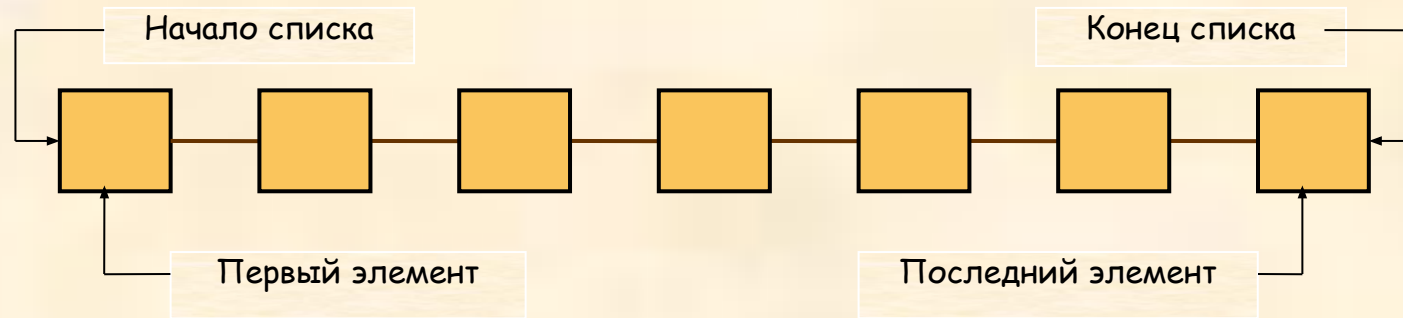
Такой подход позволяет сосредоточиться на сущности задачи и способах её решения.

Вопросы описания задачи средствами языка программирования переносятся на этап реализации.

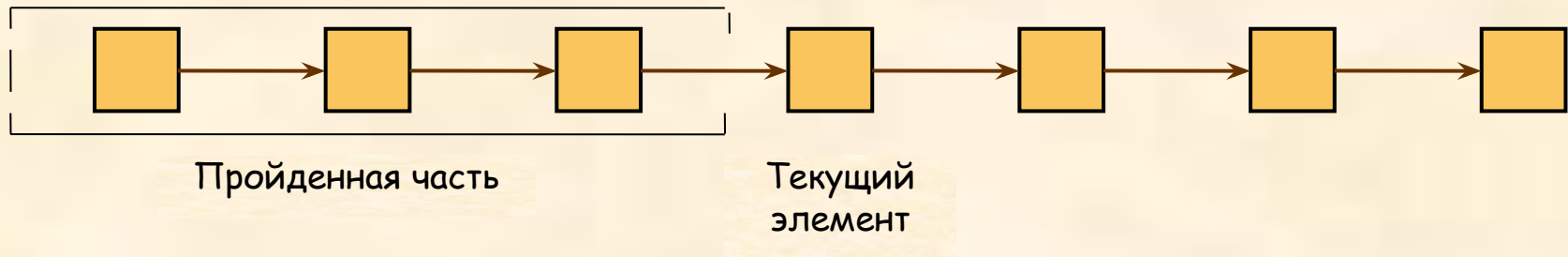
Идеи абстракции данных нашли своё отражение и в **объектно-ориентированных языках программирования**



Схематичное (модельное) представление линейного списка



Модель Л1-списка (выделен текущий элемент)



Л1-список как абстрактный тип данных

1. <i>Create</i> : $\rightarrow L_list$	Начать работу (создать пустой список)
2. <i>Null</i> : $L_list \rightarrow Boolean$	Список пуст
3. <i>Empty</i> : $L_list \rightarrow L_list$	Сделать список пустым
4. <i>GoBOL</i> : $L_list \rightarrow L_list$	Встать в начало списка (сделать текущим первый элемент)
5. <i>EOList</i> : $L_list \rightarrow Boolean$	Конец списка (не пройденная часть списка пуста)
6. <i>GoNext</i> : $L_list \rightarrow L_list$	Перейти к следующему элементу; если текущим был последний, то новое состояние списка - <i>EOList</i> ; отказ: в состояниях <i>Null</i> , <i>EOList</i>



Л1-список как абстрактный тип данных

7. <i>GetEl</i> : $L_list \rightarrow El$	Получить текущий элемент; отказ: в состояниях <i>Null</i> , <i>EOList</i>
8. <i>Insert</i> : $L_list \otimes El \rightarrow L_list$	Добавить элемент перед текущим (в состоянии <i>EOList</i> добавить элемент в конец списка), сделать текущим новый элемент
9. <i>Replace</i> : $L_list \otimes El \rightarrow L_list$	Заменить текущий элемент; отказ: в состояниях <i>Null</i> , <i>EOList</i>
10. <i>Delete</i> : $L_list \rightarrow L_list$	Удалить текущий элемент, следующий сделать текущим; отказ: в состояниях <i>Null</i> , <i>EOList</i>
11. <i>Destroy</i> : $L_list \rightarrow$	Закончить работу



Представление и реализация линейных списков (например, Л1-списка):

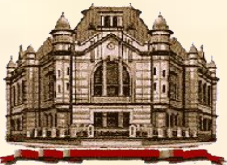
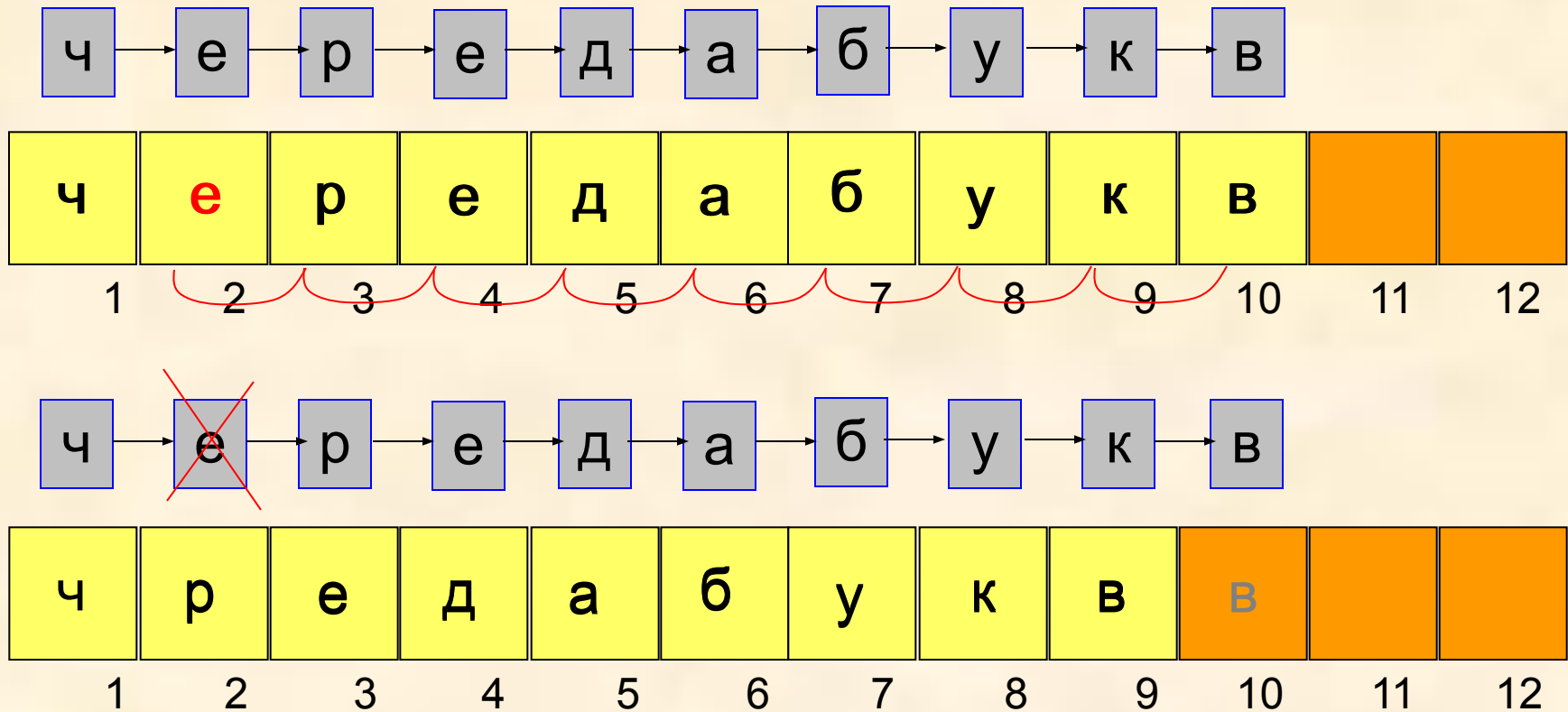
- *Непрерывная реализация*
- *Ссылочное представление в связанной (динамической) памяти*
- *Ссылочное представление на базе вектора*



Непрерывная реализация на базе вектора

Линейный список представлен одномерным массивом так, что соседние элементы списка записаны в соседние элементы массива.

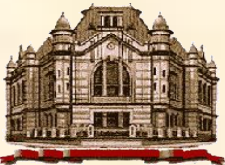
Пример операции «удалить элемент списка»:



Непрерывная реализация на базе вектора

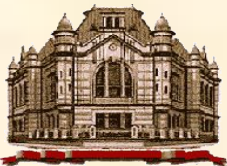
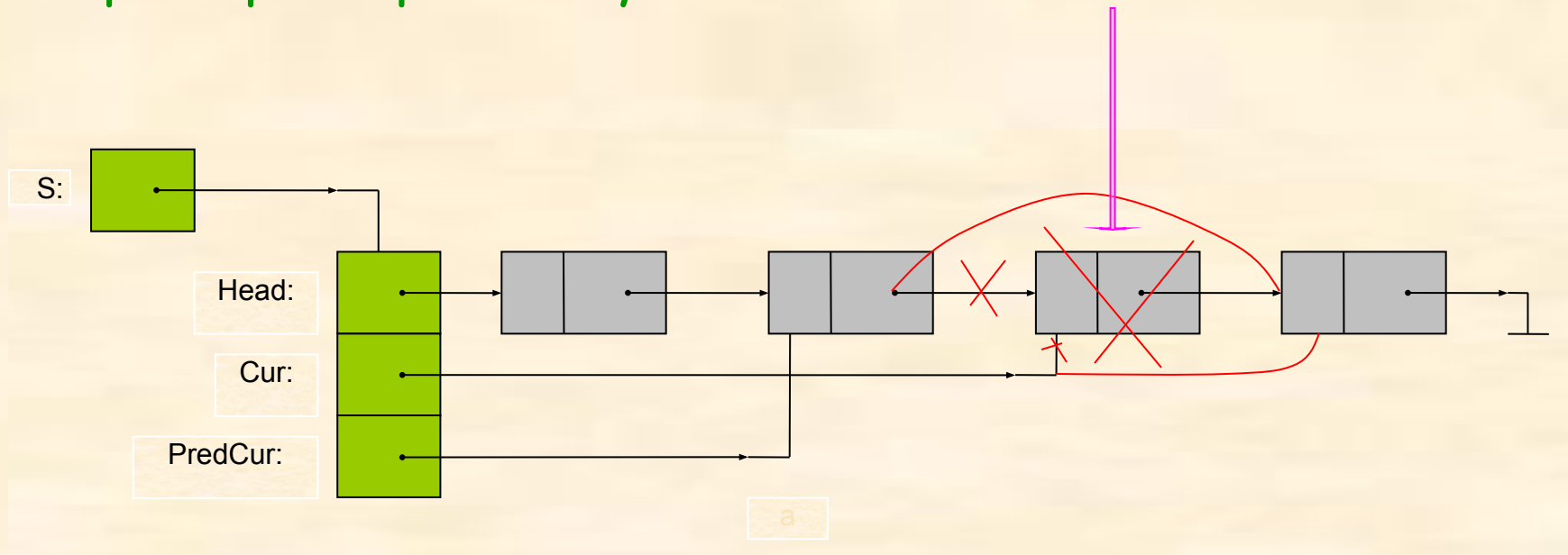
Очевидный **недостаток** - необходимость сдвига элементов массива при выполнении операций вставки и удаления элемента списка.

Операции удаления и вставки при такой реализации являются **массовыми**, т. е. требующими выполнения элементарных операций в количестве, в среднем пропорциональном числу элементов списка.



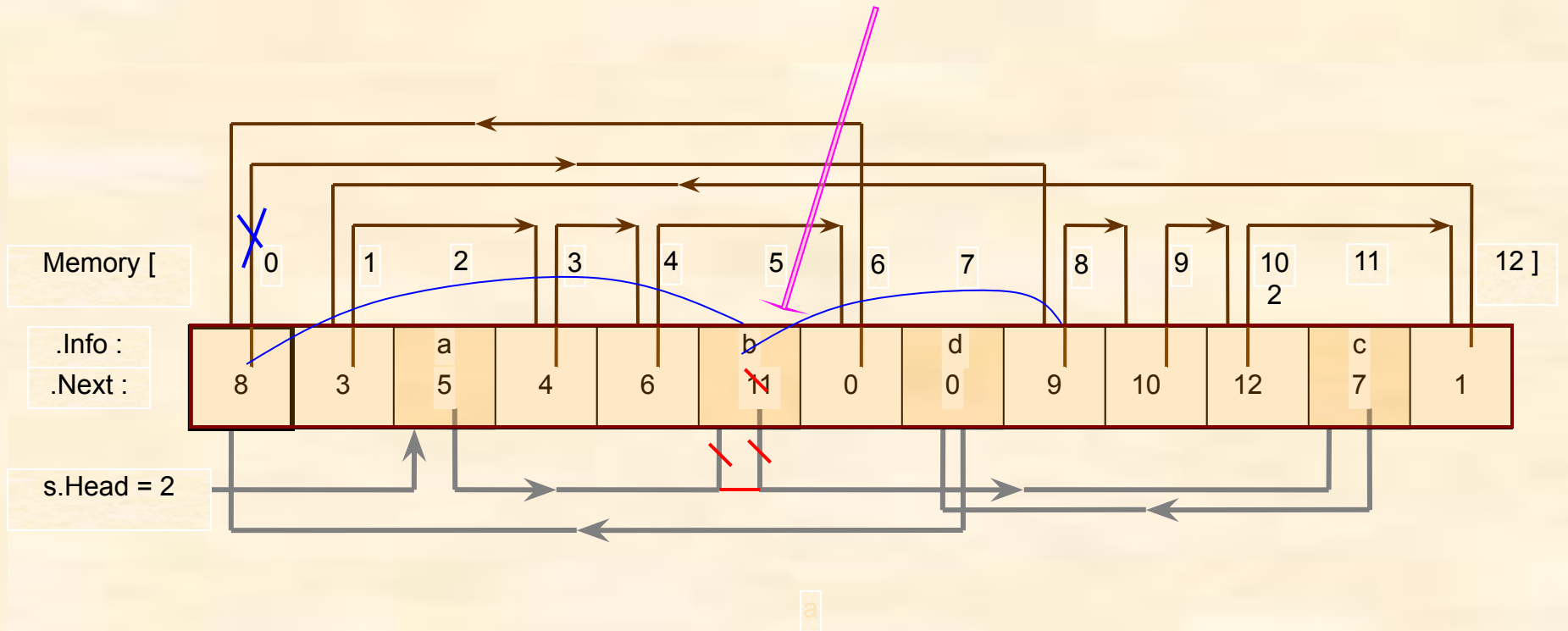
Ссылочное представление в связанной (динамической) памяти

Пример операции «удалить элемент списка»:



Ссылочное представление на базе вектора

Пример операции «удалить элемент списка»:



Литература

- Алексеев А. Ю., Ивановский С. А., Куликов Д. В. **Динамические структуры данных**: Учеб. пособие. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2004.
- 1. Ивановский С.А., Калмычков В. А., Лисс А. А., Самойленко В.П. **Представление и обработка структурированных данных**: Практикум по программированию / СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2002.



Конец вводной части

Далее

Рекурсивная обработка линейных списков



Литература

Основная

Алексеев А. Ю., Ивановский С. А., Куликов Д. В.
Динамические структуры данных: Учеб. пособие. СПб.:
Изд-во СПбГЭТУ «ЛЭТИ», 2004. (с. 20)

Дополнительная

- Алагич С., Арбиб М. Проектирование корректных структурированных программ. М.: Радио и связь, 1984.
- Хендерсон П. Функциональное программирование. Применение и реализация. М.: Мир, 1983.
- Бауэр Ф.Л., Гооз Г. Информатика. Вводный курс: В 2 ч.- М.:Мир, 1990.
- Фостер Дж. Обработка списков. М.: Мир, 1974

Учебный курс МТИ

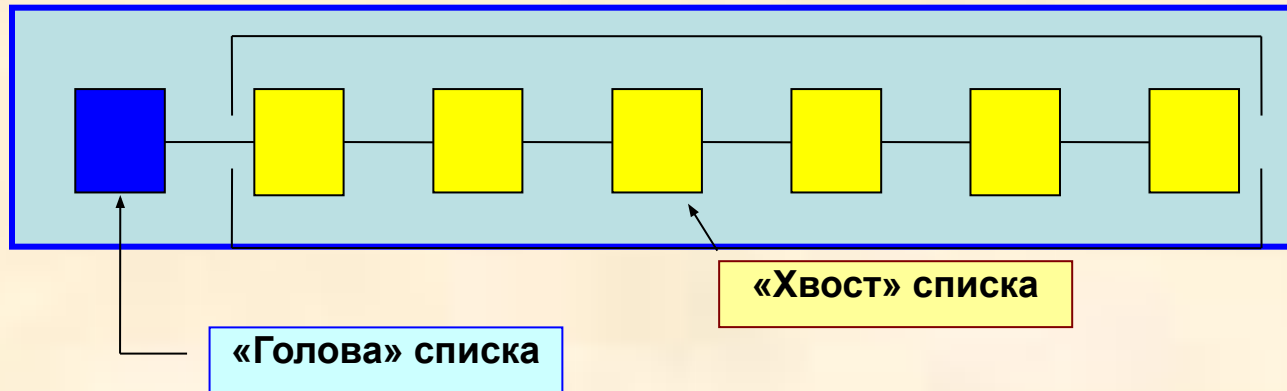
Абельсон Х., Сассман Д.Д.,
Сассман Д.

Структура и интерпретация
компьютерных программ - М.:
Добросвет, КДУ, 2006



Массачусетский технологический институт ([англ. Massachusetts Institute of Technology, MIT](#)) — [университет](#) и [исследовательский центр](#), расположенный в [Кембридже](#) (шт. [Массачусетс](#), [США](#)). Иногда также упоминается как *Массачусетский институт технологий* и *Массачусетский технологический университет*.

Модельное представление линейного списка



Скобочная запись:

$x = (a\ b\ c\ d\ e)$ или $[a, b, c, d, e]$ или $(a; b; c; d; e)$

Функции-селекторы: «голова» - *head, first*
«хвост» - *tail, rest*

применяются к непустому списку и позволяют разобрать его на составные части.

Например, $Head(x) = a$, $Tail(x) = (b\ c\ d\ e)$,
 $Head(Tail(x)) = b$, $Tail(Tail(x)) = (c\ d\ e)$

Функция-конструктор $Cons(x, y)$

$x = a$ - элемент базового типа,

$y = (b\ c\ d\ e)$ - список

$$Cons(x, y) = (a\ b\ c\ d\ e)$$

СВОЙСТВА:

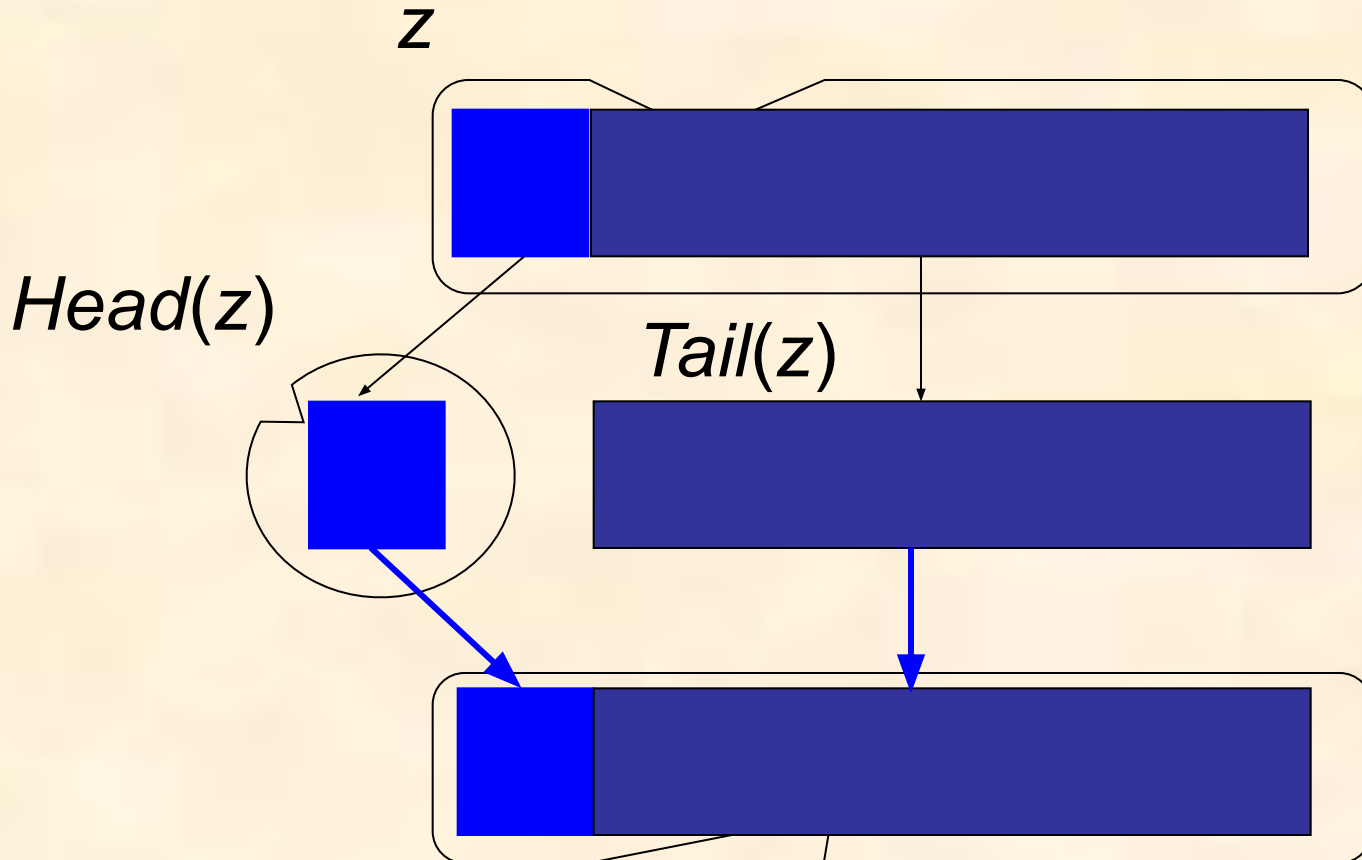
$$Cons(Head(z), Tail(z)) = z$$

$$Head(Cons(x, y)) = x$$

$$Tail(Cons(x, y)) = y$$

Иллюстрация

$Cons(Head(z), Tail(z)) = z$



$$Cons(Head(z), Tail(z)) = z$$

Функция-конструктор $Cons(x, y)$

Пустой список: $()$ или Nil

$$Cons(a, Nil) = Cons(a, ()) = (a)$$

$$(a\ b\ c\ d\ e) =$$

$$Cons(a, Cons(b, Cons(c, Cons(d, Cons(e, Nil))))))$$

Это «операционное» представление списка
(формальное определение скобочного представления -
далее)

Функция-индикатор: $Null(z)$

$$Null(Nil) = true, \quad z = (a\ b\ c\ d\ e) \rightarrow Null(z) = false$$

$$\text{Всегда } Null(Cons(x, y)) = false$$

Примеры

Пример 1.1. Функция *Size*, вычисляющая число элементов (длину) списка.

<i>y</i>	<i>Size(y)</i>
(d c b a)	4
(f)	1
<i>Nil</i>	0

Случай 1: $y = Nil$ $Size(y) = 0$

Случай 2: $y \neq Nil$ пусть $Size(Tail(y)) = n$,

тогда $Size(y) = n + 1$.

Рекурсивная функция

$Size(y) = \text{if } Null(y) \text{ then } 0 \text{ else } Size(Tail(y)) + 1.$

Примеры

Пример 1.2. Функция *Sum*, вычисляющая сумму элементов числового списка.

<i>y</i>	<i>Sum(y)</i>
(2 2 10 1)	15
(-7 7)	0
<i>Nil</i>	0

Случай 1: $y = Nil$ $Sum(y) = 0$

Случай 2: $y \neq Nil$ пусть $Sum(Tail(y)) = s$,
тогда $Sum(y) = Head(y) + s$.

Рекурсивная функция

$Sum(y) = \text{if } Null(y) \text{ then } 0$
 $\text{else } Head(y) + Sum(Tail(y)).$

Пример 1.3.

Число вхождений $Numb(x, y)$ элемента x в список y

x	y	$Numb(x, y)$
b	(a b c d)	1
a	(a b b a)	2
e	(a b b a)	0
a	<i>Nil</i>	0

Случай 1: $y = Nil$ $Numb(x, y) = 0$

Случай 2: $y \neq Nil$ пусть $Numb(x, Tail(y)) = n$, тогда

подслучай 2.1: $x = Head(y) \rightarrow Numb(x, y) = n + 1$

подслучай 2.2: $x \neq Head(y) \rightarrow Numb(x, y) = n$

Рекурсивная функция

```
Numb ( x, y) = if Null (y) then 0
              else if x = Head (y) then Numb (x, Tail (y)) + 1
              else Numb (x, Tail (y))
```


Пример 1.4.

Функция *Concat*, соединяющая два списка в один.

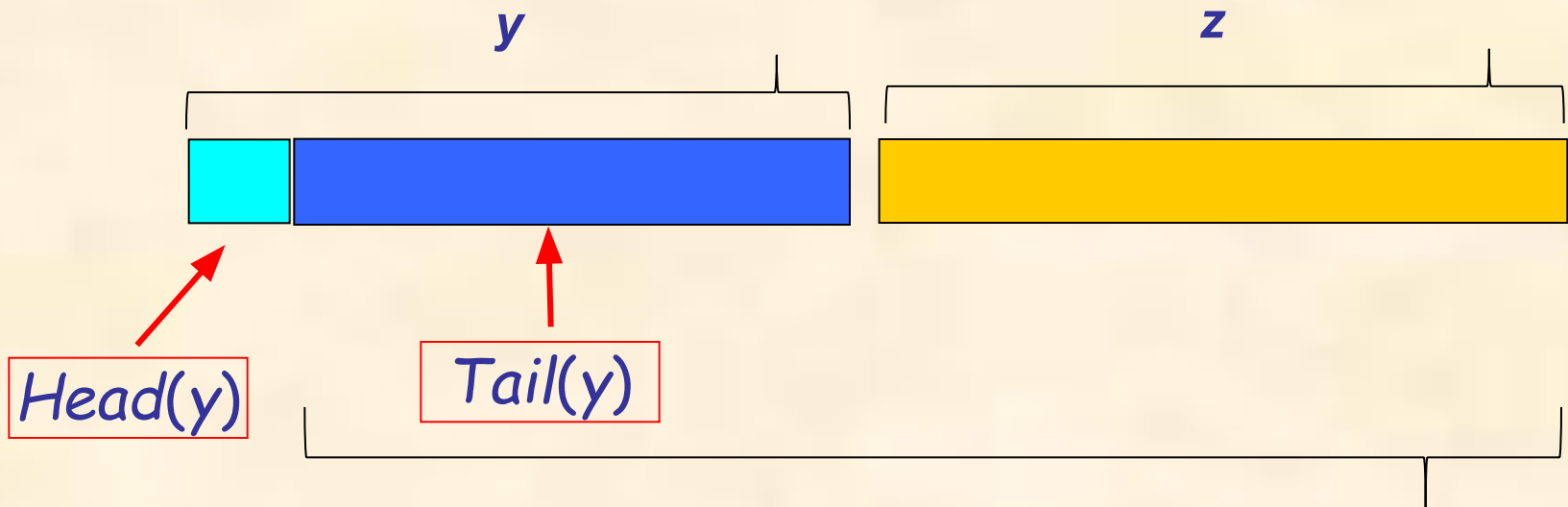
Например, $Concat(y, z) = (a\ b\ c\ d)$ для $y = (a\ b)$ и $z = (c\ d)$.

Случай 1: $y = Nil$ $Concat(y, z) = z$

Случай 2: $z = Nil$ $Concat(y, z) = y$

Случай 3: $y \neq Nil$ и $z \neq Nil$ пусть $Concat(Tail(y), z) = u$,

тогда $Concat(y, z) = Cons(Head(y), u)$



$Concat(Tail(y), z) = u$

Рекурсивная функция

$Concat(y, z) = \text{if } Null(y) \text{ then } z$
 $\text{else } Cons(Head(y), Concat(Tail(y), z)).$

Пример

Ср. с определением
на сл.25

$Concat((a\ b), (c\ d)) = Cons(a, \underline{Concat((b), (c\ d))}).$

~~$Concat((b), (c\ d)) = Cons(b, \underline{Concat(Nil, (c\ d))}).$~~

~~$Concat(Nil, (c\ d)) = (c\ d).$~~

~~$Concat((b), (c\ d)) = \leftarrow Cons(b, (c\ d)) = (b\ c\ d).$~~

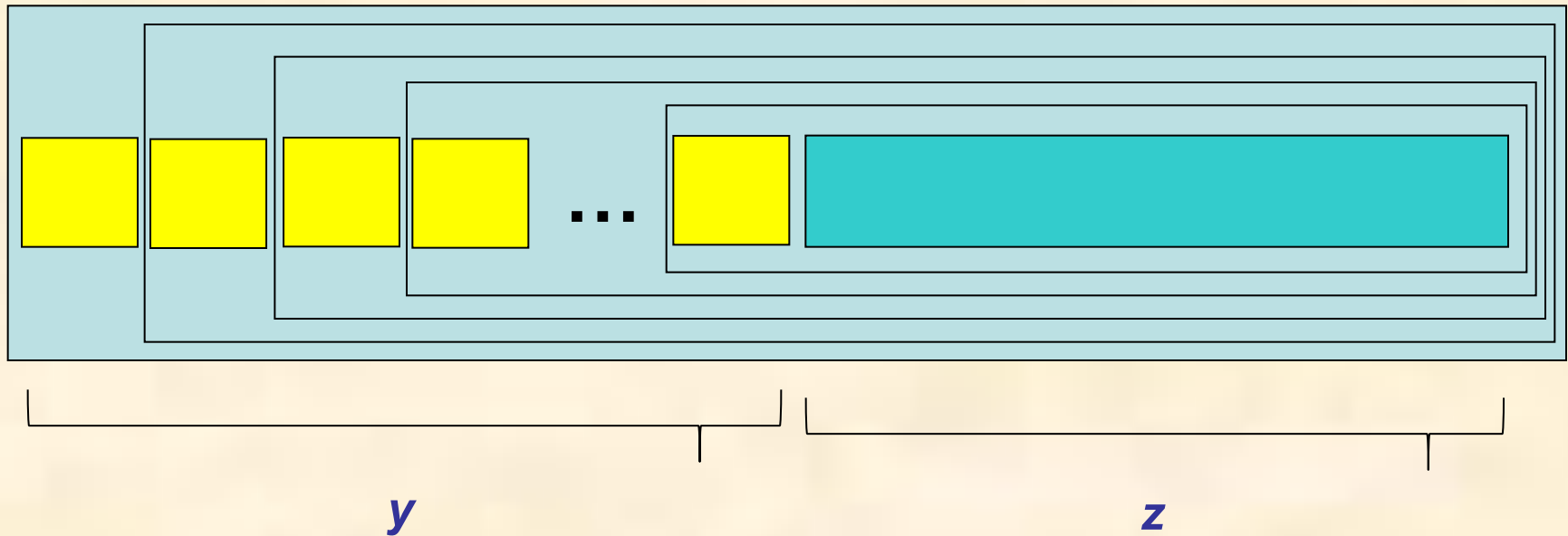
~~$Concat((a\ b), (c\ d)) = Cons(a, (b\ c\ d)) = \underline{(a\ b\ c\ d)}.$~~

Замечания: 1. Список y разбирается и затем собирается даже если список z пуст.

2. Функция $Cons$ вызывается $Size(y)$ раз.

Разборка и сборка при выполнении функции

Concat(y, z)



Пример 1.5. Функция *Append*, добавляющая элемент x в конец списка y .

Например, $Append(y, x) = (a\ b\ c)$ для $y = (a\ b)$ и $x = c$.

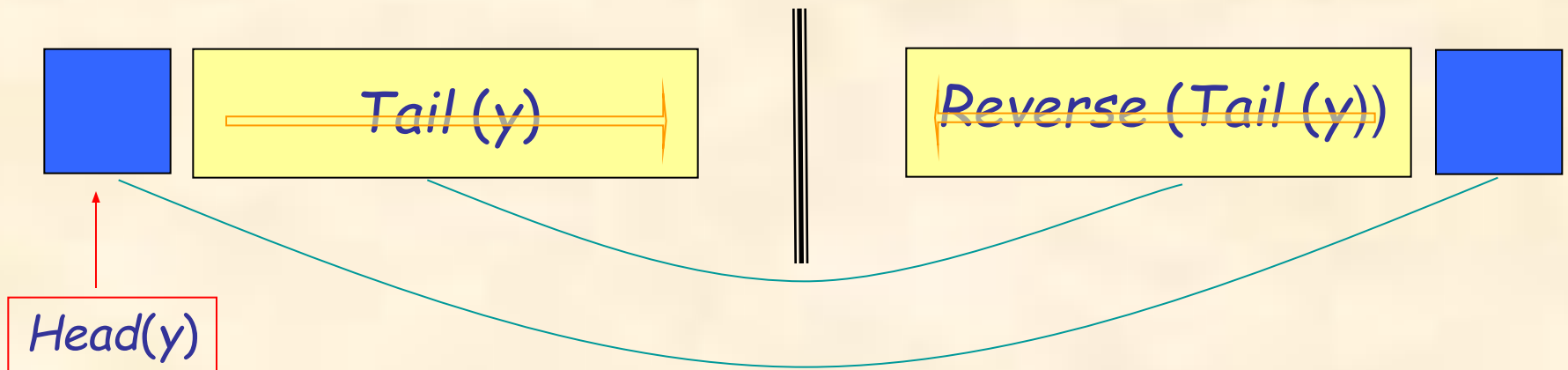
$Append(y, x) = Concat(y, Cons(x, Nil))$.



Пример 1.6. Функция *Reverse*, обращающая список.

Например, $Reverse((a\ b\ c)) = (c\ b\ a)$.

$Reverse(y) = \text{if } Null(y) \text{ then } Nil$

$\text{else } Concat(Reverse(Tail(y)), Cons(Head(y), Nil))$.



Вызовы	Возвращаемые значения
 <pre> Reverse((a b)) Concat(Reverse((b)), Cons(a, Nil)) * Reverse((b)) Concat(Reverse(Nil), Cons(b, Nil)) * Reverse(Nil) * Cons(b, Nil) * Cons(a, Nil) Cons(b, Concat(Nil, (a))) * Concat(Nil, (a)) </pre>	<pre> (b a) (b a) (b) (b) Nil (b) (a) (b a) (a) </pre> 

Примечание: На рисунке не отражены вызовы селекторов Head и Tail. Вместо этого на соответствующие места подставлены возвращаемые ими значения.

2 последних строки – это раскрытие операции второй строки.

$Concat(y, z) = Cons(Head(y), Concat(Tail(y), z))$, где
 Y - это $Reverse(b)$, значение которого уже известно и равно (b)
 Z - $Cons(a, Nil)$, значение (a)

Сложность функции *Reverse* (начало)

$Concat(y, z) =$
if $Null(y)$ then z
else $Cons(Head(y), Concat(Tail(y), z))$.

Количество вызовов $C(n)$ функции $Cons$ в
 $Concat$,

где $n = Size(y)$

$$C(n) = C(n-1) + 1; \quad C(0) = 0$$

$$\rightarrow C(n) = n$$

Сложность функции *Reverse* (продолжение)

Reverse(*y*) = if *Null* (*y*) then *Nil*
else *Concat* (*Reverse* (*Tail* (*y*)),
Cons (*Head* (*y*), *Nil*)).

Количество вызовов $R(n)$ функции *Cons* в *Reverse*

$$R(n) = R(n-1) + 1 + C(n-1); \quad R(0) = 0$$

$$\rightarrow R(n) = R(n-1) + 1 + (n-1) = R(n-1) + n;$$

→ Метод итераций:

$$\begin{aligned} \rightarrow R(n) &= R(n-1) + n = R(n-2) + (n-1) + n = \\ &= R(n-3) + (n-2) + (n-1) + n = \dots \end{aligned}$$

$$\rightarrow R(n) = n + (n-1) + (n-2) + \dots + 2 + 1 = n(n+1)/2$$

$$\text{Итак, } R(n) = n(n+1)/2$$

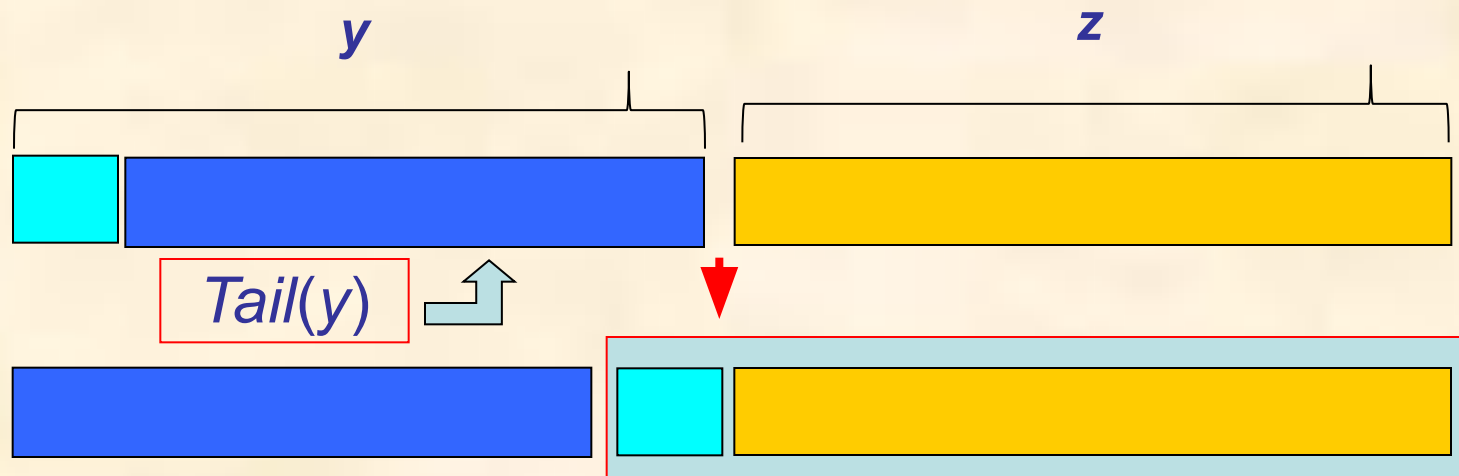
Пример 1.7.

Функция *Reverse2*, обращающая список.

Введем вспомогательную функцию *Rev*, второй параметр которой используется как «накапливающий»:

$Rev(y, z) = \text{if } Null(y) \text{ then } z$
 $\text{else } Rev(Tail(y), Cons(Head(y), z));$

Тогда $Reverse2(y) = Rev(y, Nil)$.



Пример. *Reverse2* (*y*) при *y* = (*a b*)

ВЫЗОВЫ	Возвращаемые значения
<i>Reverse2</i> ((<i>a b</i>))	(<i>b a</i>)
<i>Rev</i> ((<i>a b</i>), <i>Nil</i>)	(<i>b a</i>)
<i>Rev</i> ((<i>b</i>), <i>Cons</i> (<i>a</i> , <i>Nil</i>))	(<i>b a</i>)
* <i>Cons</i> (<i>a</i> , <i>Nil</i>)	(<i>a</i>)
<i>Rev</i> (<i>Nil</i> , <i>Cons</i> (<i>b</i> , (<i>a</i>)))	(<i>b a</i>)
* <i>Cons</i> (<i>b</i> , (<i>a</i>)))	(<i>b a</i>)

Количество вызовов конструктора *Cons* при обращении функцией *Reverse2* списка длины *n*.

$$Q(n) = Q(n - 1) + 1,$$

$$Q(0) = 0$$

$$\rightarrow Q(n) = n \quad (!)$$

Ср. с $R(n) = n(n + 1)/2$

Определение скобочной записи линейного списка

$\langle L_list(EI) \rangle ::= \langle Null_list \rangle \mid \langle Non_null_list(EI) \rangle$

$\langle Null_list \rangle ::= Nil$

$\langle Non_null_list(EI) \rangle ::= \langle Pair(EI) \rangle$

$\langle Pair(EI) \rangle ::= (\langle Head_I(EI) \rangle . \langle Tail_I(EI) \rangle)$

$\langle Head_I(EI) \rangle ::= \langle EI \rangle$

$\langle Tail_I(EI) \rangle ::= \langle L_list(EI) \rangle$

Обозначения

Здесь $L_list(EI)$ - линейный список из элементов типа EI ,
 $Null_list$ - пустой список,
 $Non_null_list(EI)$ - непустой линейный список,
 $Head_I(EI)$ - «голова» списка,
 $Tail_I(EI)$ - «хвост» списка,
 $Pair(EI)$ - упорядоченная пара «голова»-«хвост», т. е.
элемент декартова произведения.

Терминальные символы:

- Nil обозначает пустой список,
- $(.)$ использованы для обозначения элемента декартова произведения.

Примеры

Полный вид	Сокращенная запись	Вариант
$(a . (b . (c . (d . Nil))))$	$(a b c d)$	*
$(d . Nil)$	(d)	*
Nil	Nil	$()$

$(a . (b . (c . (d . \underline{Nil})))) \rightarrow (a . (b . (c . (d \underline{ }))))$

$(a . (b . (c \underline{ } (d \underline{ })))) \rightarrow (a . (b \underline{ } (c d \underline{ })))$

$(a \underline{ } (b c d \underline{ })) \rightarrow (a b c d)$

Базовые функции:

- индикатор *Null* (предикат: список пуст),
- селекторы *Head* («голова» списка) и *Tail* («хвост» списка),
- конструктор *Cons*

0) *Nil*: $\rightarrow \text{Null_list}$;

1) *Null*: $L_list(EI) \rightarrow \text{Boolean}$;

2) *Head*: $\text{Non_null_list}(EI) \rightarrow EI$;

3) *Tail*: $\text{Non_null_list}(EI) \rightarrow L_list(EI)$;

4) *Cons*: $EI \otimes L_list(EI) \rightarrow \text{Non_null_list}(EI)$;

Функциональная
спецификация
линейного списка

$f(x)$ или $f()$ обозначено $f: A \rightarrow B$

$f(x, y)$ или $f(,)$ обозначено $f: A \otimes B \rightarrow C$

Система правил (аксиом) A1-A5

A1) $Null (Nil) = true;$

A2) $Null (Cons (x, y)) = false;$

A3) $Head (Cons (x, y)) = x;$

A4) $Tail (Cons (x, y)) = y;$

A5) $Cons (Head (z), Tail (z)) = z.$

для всех x типа $E1$,

для всех y типа $L_list (E1)$,

для всех z типа $Non_null_list (E1)$

Использование функции *Cons* для конструирования списков

$(a) = (a . Nil) = Cons (a, Nil);$

$(a b c) = (a. (b. (c. Nil))) =$

$Cons (a, Cons (b, Cons (c, Nil))).$

Построение каждой «точечной» пары (значения типа *Pair (E1)*) требует однократного применения конструктора *Cons*.

Определение типа *Pair (E1)*, не связанное с конкретным (скобочным) представлением :

$\langle Pair (E1) \rangle ::= Cons (\langle Head_1 (E1) \rangle , \langle Tail_1 (E1) \rangle)$

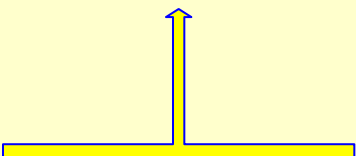
Это «операционное» представление списка

Правило A5 становится при этом излишним и может быть выведено из аксиом. A3, A4 (проверить!).

К правилу A5

Пусть $z = \text{Cons}(x, y)$.

Тогда A5 есть $\text{Cons}(\text{Head}(z), \text{Tail}(z)) =$
 $= \text{Cons}(\text{Head}(\text{Cons}(x, y)), \text{Tail}(\text{Cons}(x, y))) =$


 $= \text{Cons}(\text{по } A3: x, \text{по } A4: y)$

Т.е. A5 теорема, а не аксиома.

КОНЕЦ ЛЕКЦИИ

КОНЕЦ ЛЕКЦИИ

КОНЕЦ ЛЕКЦИИ

КОНЕЦ ЛЕКЦИИ

КОНЕЦ ЛЕКЦИИ

КОНЕЦ ЛЕКЦИИ

КОНЕЦ ЛЕКЦИИ

КОНЕЦ ЛЕКЦИИ