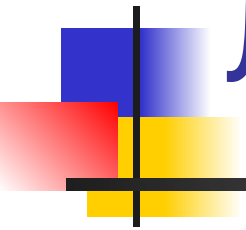
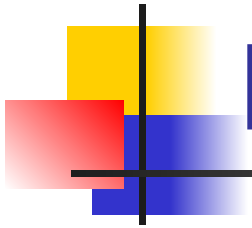


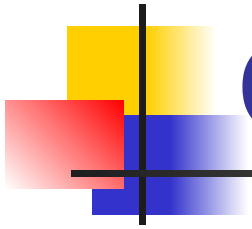
Лекция

RAISE Specification Language:
списки и операции со списками



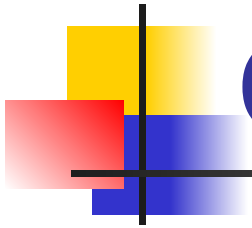
План лекции

- Списки. Свойства списков
- Описание типов
- Литералы и агрегаты
- Операции со списками
- Диаграмма Гогена
- Пример «хорошего стиля»



Списки. Свойства списков

- каждый элемент может встретиться несколько раз
- порядок определен (и существенен)



Описание типов

type

LT1 = T1**-list** КОНЕЧНЫЕ СПИСКИ

LT2 = T1*

NLT1 = T1**-inflist** БЕСКОНЕЧНЫЕ СПИСКИ

NLT2 = T1+

Литералы и агрегаты

<.1,2,3.>
<..>
списочное выражение (rangered list expression)
<.3...7> = <.3,4,5,6,7.>
<.3..3.> = <.3.>
<.3..2.> = <..>
генерация списка на основе имеющегося
<. 2*n | n **in** <.1..100.> **:- true** .>
<. 2*n | n **in** <.1..100.> .>

<.n | n **in** <.1..100.> **:-** is_a_prime(n).> = ...

value
INTLIST : (**Int** >< **Int**)**-list** = <. (1,2), (2,2), (2,1), (3,1).>
.....
... = <. (x,f(y)) | (x,y) **in** INTLIST **:-** x>y .>

Тип Text и списки символов

"abc" = <.'a','b','c'.> текст из трех символов

"" = <..> текст нулевой длины

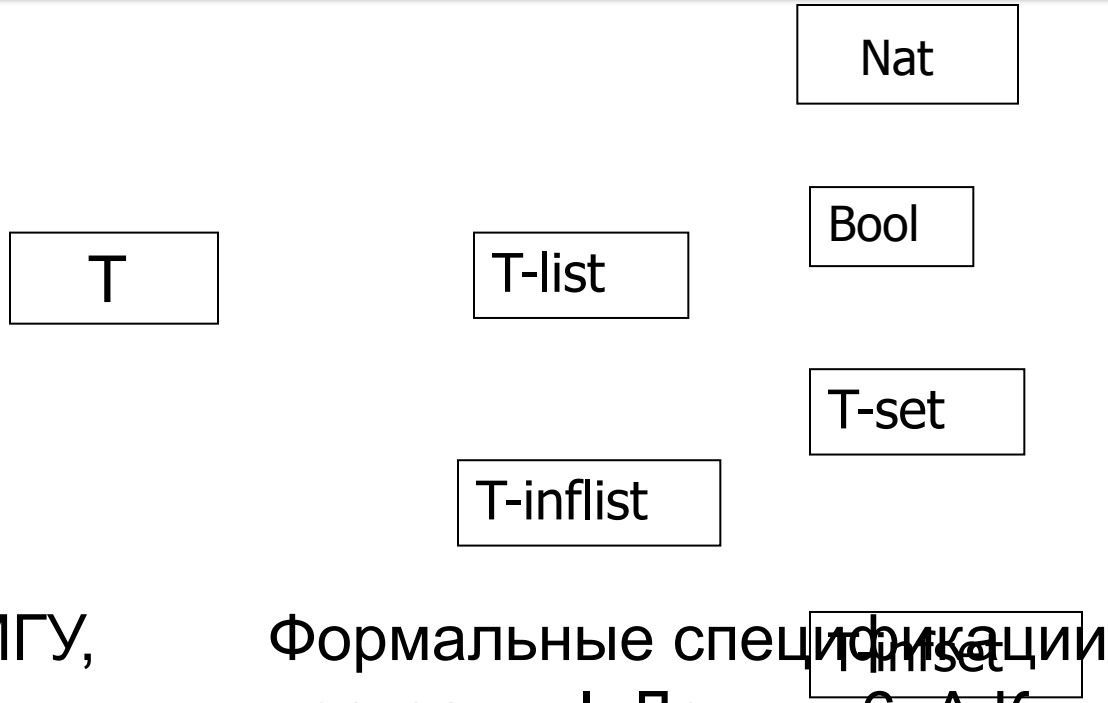
Операции со списками

^ : T-**list** >< T-**list** -> T-**list**
in : T >< T-**list**
len : T-**list** -> **Nat**
hd : T-**list** -~-> T
tl : T-**list** -~-> T-**list**
inds : T-**list** -> **Nat-set**
elems : T-**list** -> T-**set**

- Свойства операций
- **inds** fl = {1..**len** fl} для конечных списков
- **inds** il = {idx | idx : **Nat :-** idx >= 1} для бесконечных списков
- **elems** l = {l(idx) | idx : **Nat :-** idx **isin inds** l}

Диаграмма Гогена

Задание: Нарисуйте связи, которые задают операции над списками между этими типами данных



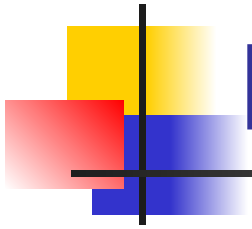
Пример: Очередь

```
QUEUE =
  class
    type
      Element,
      Queue = Element-list
    value
      empty : Queue,
      put : Element >< Queue -> Queue,
      get : Queue -~-> Queue >< Element
    axiom forall e : Element, q : Queue
      empty is <..>,
      put (e,q) is q ^ <.e.>,
      get(q) is (tl q, hd q)
      pre q ~= empty
  end
```

Пример: Сортировка (1)

Цель примера – показать “хороший стиль”:
мы не описываем алгоритмы,
мы описываем только свойства результата.

```
LIST_PROPERTIES =
  class
    value
      is_permutation : Int-list >< Int-list -> Bool,
      is_sorted : Int-list -> Bool
    axiom forall l,l1,l2 ; Int-list :-
      is_permutation(l1,l2) is
        (all i : Int :-
           card {idx | idx : Nat :- idx isin indx l1 /\ l1(idx) = i} =
           card {idx | idx : Nat :- idx isin indx l2 /\ l2(idx) = i},
      is_sorted(l) is
        (all idx1,idx2 : Nat :-
          {idx1,idx2} <<= inds l /\ idx1 < idx2 =>
           l(idx1) <= l(idx2))
end
```



Пример: Сортировка (2)

```
SORTING =
  extend LIST_PROPERTIES with
  class
    value
      sort : Int-list -> Int-list
    axiom forall l : Int-list :-

      sort(l) as l1 post is_permutation(l1,l2) /\ is_sorted(l1)
end
```

Замечание. В данном примере мы разбили спецификацию на два модуля. В первом собраны вспомогательные определения. Второй модуль описывает функциональность собственно функции sort.