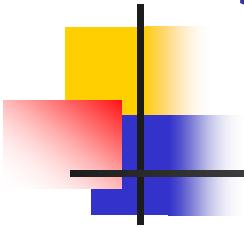
Формальная спецификация и верификация программ. Лекция 2. Виды формальных спецификаций

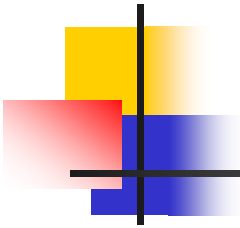
А.К.Петренко, А.В.Хорошилов,
Е.В.Корныхин
МГУ ВМК, ИСП РАН
http://sp.cmc.msu.ru/courses/fmsp

Осень, 2012

Элементы Оснований разработки программ

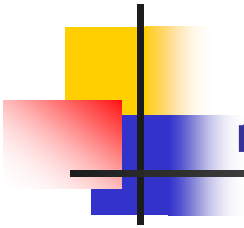


- Разделение понятий
  - Реализации программы (алгоритма) и
  - Поведения программы
- Абстракция
  - Виды абстракции
  - Методы изменения уровня абстракции и сопоставления уровней



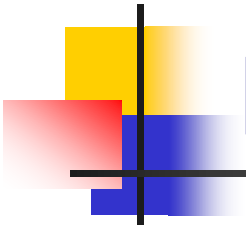
Формальные методы

- Спецификация (поведения)
- Языки формальной спецификации
- Инструменты поддержки
  - Анализ корректности спецификации
  - Анализ соответствия спецификации и реализации (верификация)
    - Анализ соответствия разных уровней абстракции, разных моделей
  - Отслеживание связей с требованиями, функциями реализации, тестами и др.



Для чего нужны спецификации

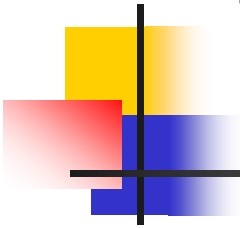
- Помощь в дизайне (уточнить треб, вскрыть открытые вопросы и противоречия в треб/проекте архитектуры
- Зафиксировать проектные решения для передачи знаний (целевой продукт реверс-инженерии)
- Альтернативная информационная структура для анализа системы (корректность, сравнение полноты с реализацией, критерии качества/полноты верификации)
- «Мечта/идеал» – генерация программ по спецификации
  - (не всегда целевых программ, например, тестов, доказательств (evidence))



Где нужны спецификации

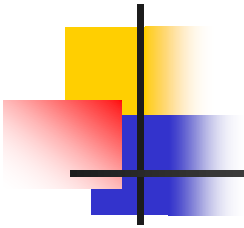
- Там, где нет готовых шаблонных решений
- Большой круг разработчиков/пользователей => задача фиксации и передачи знаний о системе
- Ответственное SW/HW, задачи надежности и отслеживания требований

Зачем нужны языки спецификаций?



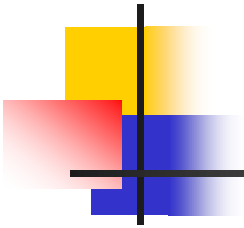
- Можно ли рассматривать программу на Java или Си как спецификацию ?
- В чем плюсы введения специальных языков спецификации?
- В чем минусы введения языков спецификации?

Пример: «Очередь» - Неформальное описание системы

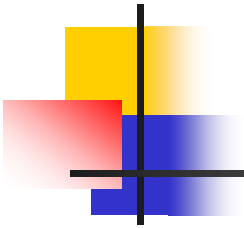


- Требуется спроектировать систему, реализующую операции для работы с «очередями»
- Система будет предоставлять программный интерфейс (ПИ, или Application Program Interface - API)

Первый шаг формализации. Выбор терминов (денотатов)



- Типы данных
  - Очередь - Queue
  - Элемент очереди - Element
- Операции
  - Поставить в очередь - append
  - Кого обслуживать следующим? - first
  - Кто останется после этого? - rest
- Константа
  - Пустая очередь - empty



Известные виды спецификаций

- Алгебраические/аксиоматические
- Явные/исполнимые/алгоритмические
- Неявные/неисполнимые/ограничения

Сигнатура (структура) интерфейса - простейший вид спецификации

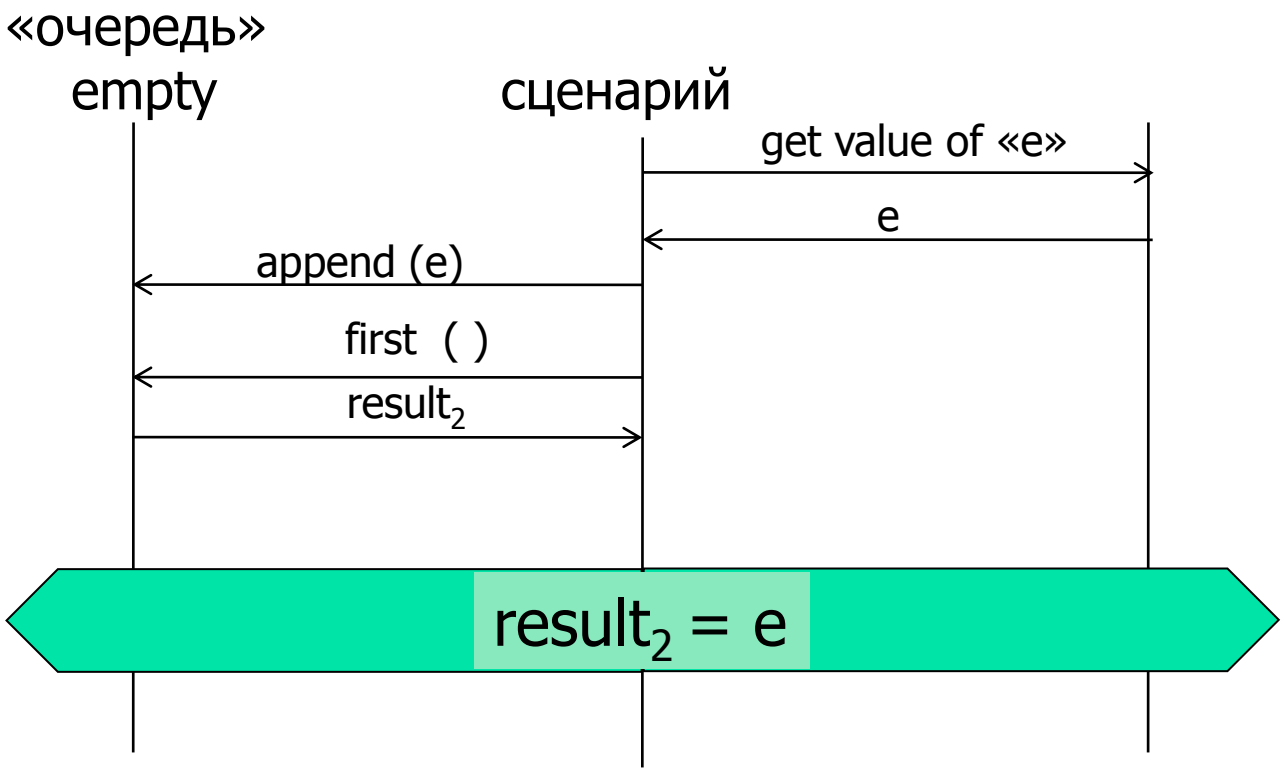
```
scheme QUEUE =  class
type        Element,
            Queue
value       empty : Queue,
            append : Element >< Queue -> Queue,
            first : Queue -~-> Element,
            rest : Queue -~-> Queue
end
```

Мы описали структуру, то есть набор элементов и связи между элементами.

Очередь. Алгебраическая спецификация на RSL

```
scheme QUEUE = class
type            Element,
                Queue
value           empty: Queue,
                append : Element >< Queue -> Queue,
                first : Queue -~-> Element,
                rest : Queue -~-> Queue
axiom forall e : Element, q : Queue :-
                empty ~= append (e,q),
                first(append(e, empty)) is e,
                rest(append(e,empty)) is empty,
                first(append(e, q)) is if q=empty then e
                        else first(q) end,
                rest(append(e,q)) is if q=empty then empty
                        else append(e, rest(q)) end
end
```

Интерпретация аксиомы first(append(e, empty)) **is** e



Очередь. Явная спецификация на RSL

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

Очередь. Неявная спецификация на RSL

```
scheme QUEUE = class
type            Element,
                Queue = Element-list
value           empty: Queue,
                append : Element >< Queue -> Queue,
                first : Queue -~-> Element,
                rest : Queue -~-> Queue
axiom forall e : Element, q : Queue :-
                empty is <..>,
                append(e,q) as q1 post q ^ <.e.> = q1,
                first(q) as  e
                post hd q= e pre q~=<..>,
                rest(q) as q1
                post exists e : Element :- <.e.> ^ q1 = q
                pre q~=<..>
end
```

Очередь. Явная спецификация с состоянием на RSL

```
scheme QUEUE = class
type            Element,
                Queue = Element-list
variable        queue : Queue := empty
value           empty: Queue,
                append : Element  -> write queue Queue,
                first : Unit  -~-> read queue Element,
                rest : Unit -~-> read queue Queue
axiom forall e : Element :-
                empty is <..>,
                append(e) is (queue := queue ^ <.e.>;queue),
                first() is hd queue pre queue~=<..>,
                rest() is tl queue pre queue~=<..>
end
```

Очередь. Неявная спецификация с состоянием

```
scheme QUEUE = class
type            Element,
                Queue = Element-list
variable        queue : Queue := empty
value           empty: Queue,
                append : Element  -> write queue Queue,
                first : Unit  -~-> read queue Element,
                rest : Unit -~-> read queue Queue

axiom forall e : Element :-
                empty is <..>,
                append(e) as q2 post queue = q2 /\ queue` ^ <.e.> = q2,
                first() as  e
                post hd queue = e pre queue ~=<..>,
                rest() as q2
                post exists e : Element :- <.e.> ^ q2 = queue
                pre queue~=<..>,
end
```

Очередь. Интерфейс класса на Dafny

```
class Queue<T>
{
  // create queue without elements
  constructor empty()

  // append `e' to the end of queue
  method append(e : T)

  // get first element
  method first() returns (e : T)

  // remove first element
  method rest()
}
```

Попробовать в браузере:
http://rise4fun.com/Dafny/GYsl0
(нажать “ask dafny”)

Очередь. Явная спецификация на Dafny

```
class Queue<T>
{
    var elements : seq<T>;

    constructor empty()
      modifies this;
    {
      elements := [];
    }

    method append(e : T)
      modifies this;
    {
        elements := elements + [e];
    }

    method first() returns (e : T)
        requires |elements| > 0;
    {
        return elements[0];
    }

    method rest()
        requires |elements| > 0;
        modifies this;
    {
        elements := elements[1..];
    }
}
```

Попробовать в браузере:
http://rise4fun.com/Dafny/hzIe

Очередь. Неявная спецификация на Dafny

```
class Queue<T>
{
  var elements : seq<T>;

  constructor empty()
    modifies this;
    ensures |elements| == 0;

  method append(e : T)
    modifies this;
    ensures old(elements) + [e] ==
elements;

  method first() returns (e : T)
    requires |elements| > 0;
    ensures e == elements[0];
    ensures elements ==
          old(elements); // needed?

  method rest()
     modifies this;
     requires |elements| > 0;
     ensures old(elements)[1..] ==
                    elements;
}
```

Попробовать в браузере:
http://rise4fun.com/Dafny/tuQY

Очередь. Алгебраическая спецификация на Dafny – параметризованные тесты

```
class Queue<T>
{
constructor empty()

method append(e : T)
     modifies this;

method first() returns (e : T)
   requires non_empty();

method rest()
   requires non_empty();
   modifies this;

 function non_empty() : bool
   reads this;
```

```
static method test_1(q :
Queue<T>, e : T)
     modifies q;
     requires q != null;
     requires q.non_empty();
  {
    var f1 := q.first();
    q.append(e);
    var f2 := q.first();
    assert f1 == f2;
  }
```

```
static method test_2(q :
Queue<T>, e : T)
    modifies q;
    requires q != null;
    requires !q.non_empty();
   {
     q.append(e);
     var f := q.first();
     assert f == e;
   }
}
```

Попробовать в браузере:
http://rise4fun.com/Dafny/gHI4

Очередь. Эффективная реализация на Dafny (без постусловий)

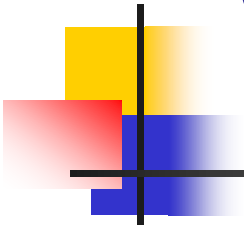
```
class Node<T>
{
  var value : T;
  var next : Node<T>;
}

// unbounded queue
class Queue<T>
{
  var head : Node<T>;
  var last : Node<T>;


  constructor empty()
    modifies this;
  {
    head := null;
    last := null;
  }
```

```
 method append(e : T)
    modifies this, (if last ==
null then {} else {last});
  {
    if (last == null) {
      last := new Node<T>;
      head := last;
    } else {
      last.next := new Node<T>;
      last := last.next;
    }
    last.value := e;
    last.next := null;
  }

method first() returns (e : T)
    requires head != null;
  {
    return head.value;
  }
```

```
method rest()
  requires head != null;
  modifies this;
{
  if (head == last) {
    head := null;
    last := null;
  } else {
    head := head.next;
  }
}


}  // end of Queue
```

В браузере:

http://rise4fun.com/Dafny/gHI4



Очередь. Эффективная реализация на Dafny (с постусловиями)

Текст программы со спецификационными аннотациями настолько большой, что не поместился на слайд. Смотреть в браузере тут:

http://rise4fun.com/Dafny/sllmp