Формальная спецификация и верификация программ.
Лекция 3.
Виды формальных спецификаций (2). Спецификации в языках программирования

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

Осень, 2012

Очередь. Алгебраическая спецификация

```
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
```

Пример трансформаций типа re-writing. Интерпретация выражений

…

axiom forall e : Element, q : Queue :-
empty ~= append (e,q),
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

rest(append(a, append (b, empty))) ⇒

append(a, rest(append (b, empty))) ⇒

append(a, empty), в свою очередь:

first(append(a, empty)) = a и
rest(append(a, empty)) = empty

Дано explicit определение функции. Написать implicit-спецификацию функции эквивалентной данной.

```
value
    f : Int ><Int >< Int -> write x, y Int
    f(a,b,c) is        if a=b then
                                y:=y+1; b*(if c>0 then x:=c; c else 0-c end)
                       else (x:=a+b;  a-b) end
```

Дается пять минут на самостоятельную попытку

Канонический процесс проектирования. Определение функции

1. Определить имя функции
2. Определить тип (сигнатуру)
   - тип аргументов
   - тип результата
   - всюду вычислимая (total) или частично вычислимая (partial)
3. Определить стиль описания
   - явное определение (explicit) – если можем предложить формулу
   - неявное определение (implicit) – если можем описать input-output отношение
   - аксиоматическое (алгебраическое) – всегда возможно

     типичное использование в случае, когда приходится использовать специфический вид аргументов (аргумент – вызов функции), например, f(g(x),x) - h(f,x)

Канонический процесс проектирования. Определение функции (2)

«Подводные камни»:

- Явная спецификация – это не реализация
- Обычно ограничиваются частичными спецификациями

1. Определить имя функции
2. Определить тип (сигнатуру)
   - тип аргументов
   - тип результата
   - всюду вычислимая (total) или частично вычислимая (partial)
3. Определить стиль описания
   - явное определение (explicit) – если можем предложить формулу
   - неявное определение (implicit) – если можем описать input-output отношение
   - аксиоматическое (алгебраическое) – всегда возможно

   типичное использование в случае, когда приходится использовать специфический вид аргументов (аргумент – вызов функции), например, f(g(x),x) - h(f,x)

Design-by-Contract

В объектно-ориентированном программировании контракт метода обычно включает следующую информацию:

- возможные типы входных данных и их значение;
- типы возвращаемых данных и их значение;
- условия возникновения **исключений**, их типы и значения;
- присутствие **побочного эффекта** метода;
- **предусловия**, которые могут быть ослаблены (но не усилены) в подклассах;
- **постусловия**, которые могут быть усилены (но не ослаблены) в подклассах;
- **инварианты**, которые могут быть усилены (но не ослаблены) в подклассах;
- *(иногда) требования/гарантии по ресурсам, например, **время** или **память**.*

Design-by-Contract (2)

В объектно-ориентированном программировании контракт метода обычно включает следующую информацию:

- возможные типы входных данных и их значение;
- типы возвращаемых данных и их значение;
- условия возникновения **исключений**, их типы и значения;

Нюансы

- присутствие **побочного эффекта** метода;
  - Как указать, что меняется в графе связанных объектов?
- **предусловия**, которые могут быть ослаблены (но не усилены) в подклассах;
- **постусловия**, которые могут быть усилены (но не ослаблены) в подклассах;
  - Принцип Барбары Лисков
  - Конструкции для «копирования» пред- и пост-условий?
  - Как писать, например, «ослабление» предусловия?
- **инварианты**, которые могут быть усилены (но не ослаблены) в подклассах;
  - Можно ли нарушать инварианты типа?
    - Случай последовательной программы
    - Случай параллельной программы
- (иногда) требования/гарантии по ресурсам, например, **время** или **память**.
  - Те же проблемы, что и в случае побочного эффекта

Design-by-Contract

- Languages that implement most DbC features natively include Cobra, D[9], Eiffel, Fortress, Lisaac, Nice, Oxygene (formerly Chrome), Racket (including higher order contracts, and emphasizing that contract violations must blame the guilty party and must do so with an accurate explanation[10]), Sather, SPARK (via static analysis of Ada programs), UniTESK, Spec#, Vala, VDM (Overture), RSL, Z, B-toolkit, UML/OCL, SDL/MSC, (?) TTCN-3, and languages that are built on top of the Microsoft .NET framework version 4.x (C#, VB.NET) through the use of the
- Library CodeContracts, Summer (Kuliamin), SystemC, OVM (System Verilog)

Требования к спецификации. Спецификация должна:

- быть простой/естественной для
  - реализатора/пользователя/ дизайнера тестов/тестировщика/ менеджера требований и инженера по качеству
- позволять проверять внутреннюю согласованность/консистентность модели и согласованность модели или реализации
- быть недвусмысленной для объективности анализа
- быть полной (определена область допустимых значений для всех функций, поведение определено при всех допустимых сценариях использования)
- не содержать лишних ограничений на реализацию

Вопрос: какие из этих требований могут быть ослаблены?
Ответ на экзамене.

Зачем нужны языки спецификаций

Вопросы:

- Можно ли рассматривать программу на Java или Си как спецификацию ?
- В чем плюсы введения специальных языков спецификации?
- В чем минусы введения языков спецификации?
- Каковы известные языки спецификаций?
- Существуют ли альтернативы языкам спецификаций?

Ответы на экзамене.

Средства спецификации в языках программирования

- Направления развития языков спецификации
  - Расширение возможностей моделирования
  - Упрощение формального анализа
  - Сближение с языками программирования
- Spec# 2003
- CodeContracts
- Ретроспектива развития языков программирования со средствами спецификации

Направления развития языков спецификации (1)

- Расширение возможностей моделирования
- Параллелизм, асинхронные взаимодействия, обработка исключений
- Временные (темпоральные) свойства
- Требования к ресурсам

Направления развития языков спецификации (2)

- Упрощение формального анализа
  - Упрощенная система типов
  - Функциональные языки (нет побочного эффекта, нет указателей)
  - Строгая семантика - как следствия:
    - Платформонезависимость
    - Классическая арифметика (не машинная)

Сближение с языками программирования: pros & cons

Pros

- Упрощение обучения
- Богатые библиотеки
- Единая система типов
- Design-by-contract

Cons

- Усложнение анализа (динамическая память)
- Размытые границы между моделью и реализацией
- Зависимость от платформы реализации

Система программирования Spec# (1)

- Spec# - язык программирования
- Компилятор
- Статический верификатор Boogie
- Spec Explorer – Система тестирования на основе моделей (формальных спецификаций)

Система программирования Spec# (2)

Главные достоинства Spec#:

- Основа популярный язык C#
- Мощный метод для спецификации и анализа (reasoning)
- Спецификация инвариантов и callbacks,
- Статические и динамические проверки

Ограничения методов (constraints)

- Предусловия
- Постусловия
- Набор исключений (throws)
- Постусловия исключений (exceptional postconditions)
- «Блоки условий» frame conditions – ограничивают часть состояния программы, которую можно модифицировать

Предусловия (1)

class *ArrayList* { **public virtual void** *Insert*(**int** *index* , **object** *value*)

requires 0 <= *index* && *index* <= *Count*;

requires !*IsReadOnly* && !*IsFixedSize*;

{ . . . }

Предусловия (2)

class *ArrayList* { **void** *Insert*(**int** *index* , **object** *value*)

requires 0 <= *index* && *index* <= *Count*

otherwise *ArgumentOutOfRangeException*;

requires !*IsReadOnly* && !*IsFixedSize*

otherwise *NotSupportedException*;

{ . . . }

Постусловия

ensures *Count* == **old**(*Count*) + 1;

ensures *value* == **this**[*index*];

ensures *Forall*{**int** *i* **in** 0 : *index* ; **old**(**this**[*i*]) == **this**[*i*]};

ensures *Forall*{**int** *i* **in** *index* : **old**(*Count*); **old**(**this**[*i*]) == **this**[*i* + 1]};

Исключения

- Два вида исключений:
  - *client failures и provider failures*.
- Client failure нарушение предусловия
- Provider failures делятся на допустимые (*admissible), например, time-out и наблюдаемые /ожидаемые ошибки (observed program errors)*.
- Клиентские и observed – unchecked exceptions

Постусловия исключений (1)

char *Read*()

throws *SocketClosedException*;

{ . . . }

Метод может возбудить исключение типа подкласса SocketClosedException и никакое другое.

Постусловия исключений (2)

void *ReadToken*(*ArrayList a*)

throws *EndOfFileException* **ensures** *a.Count* == **old**(*a.Count*);

{ . . . }

указывает, что длина *a* не изменяется, когда результатом является *EndOfFileException* .

Ограничения класса

class *AttendanceRecord* { *Student*[ ]! *students*;
bool[ ]! *absent* ;
invariant *students.Length* == *absent .Length*;

Оператор **expose** указывает, что инварианты объекта *o* временно нарушаются во время выполнения statement

expose (*o*) {
S;
}

CodeContracts (рациональные числа)

```
using System;
using System.Diagnostics .Contracts;

class Rational
{
  int numerator;
  int denominator;

  public Rational( int numerator, int denominator )
  {
    Contract.Requires( denominator ! = 0 );

    this.numerator = numerator;
    this.denominator = denominator;
  }
  public int Denominator
  {
    get
    {
      Contract.Ensures( Contract.Result<int>() == this.denominator );
      return this.denominator;
    }
  }
  [ContractInvariantMethod]
  protected void ObjectInvariant ()
  {
    Contract.Invariant ( this.denominator ! = 0 );
  }
}
```

CodeContracts (квадратный корень - 1)

- **привязка к исходному коду**
-
- **class** Math
- {
-
- **public static double** Sqrt (**double** x)
- {
- Contract.Requires(x > 0);
- Contract.Ensures (
- Contract.Result<**double**>()*Contract.Result<**double**>() == x);
-
- **...**
- }
- }

CodeContracts (квадратный корень - 2)

- **привязка через внешний интерфейс**
-
- [ContractClass(**typeof**(SqrtContract))]
- **interface** ISqrt
- {
- **double** Sqrt (**double** x);
- }
- **class** SqrtImpl : ISqrt
- {
- **double** Sqrt (**double** x) { **return** Math.Sqrt(x); }
- }
- [ContractClassFor(**typeof**(ISqrt))]
- **sealed class** SqrtContract : ISqrt
- {
- **double** Sqrt (**double** x)
- {
- Contract.Requires(x > 0);
- Contract.Ensures (
- Contract.Result<**double**>()*Contract.Result<**double**>() == x);
- **return default**(**double**);
- }
- }

CodeContracts (квадратный корень - 3)

- **варианты предусловия**
- **самый простой**
-
- Contract.Requires(x >= 0);
-
- **с исключением**
-
- Contract.Requires<IllegalArgumentException>(x >= 0);
-

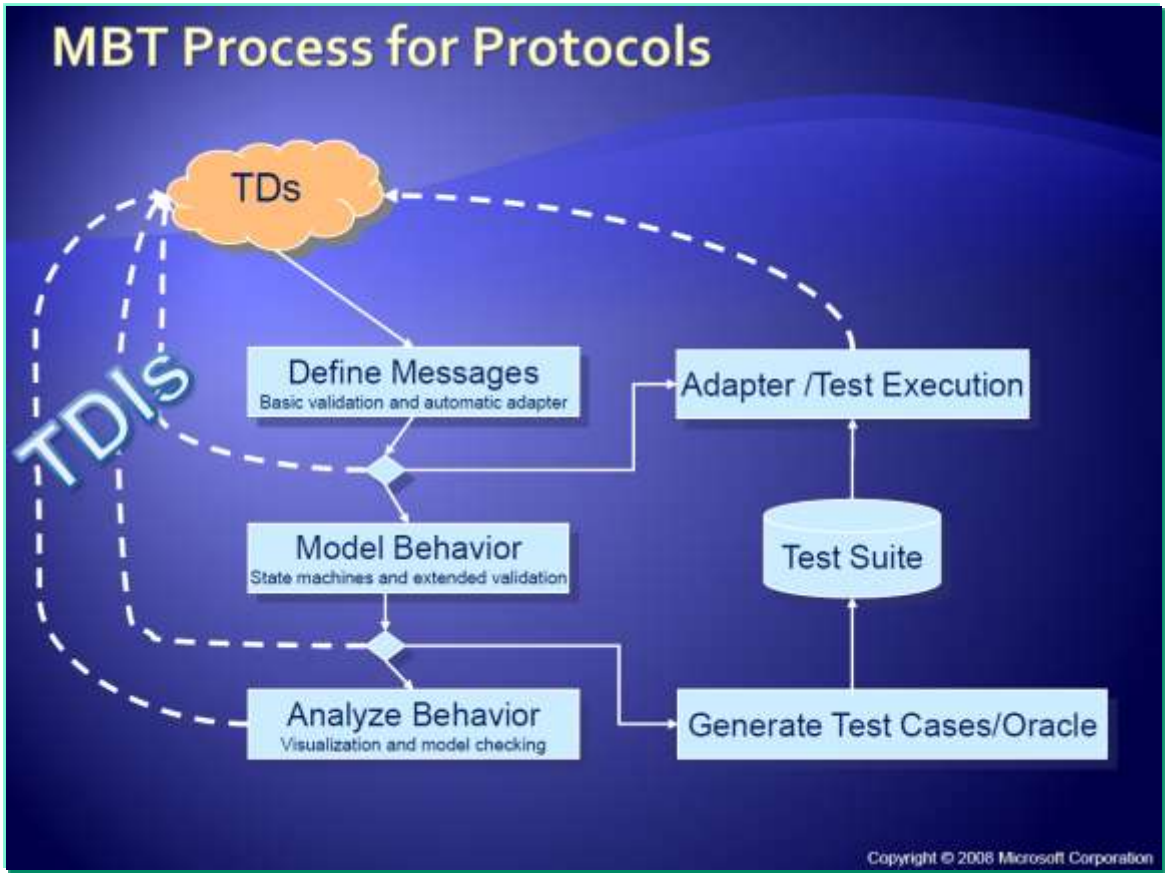
CodeContracts (квадратный корень - 4)

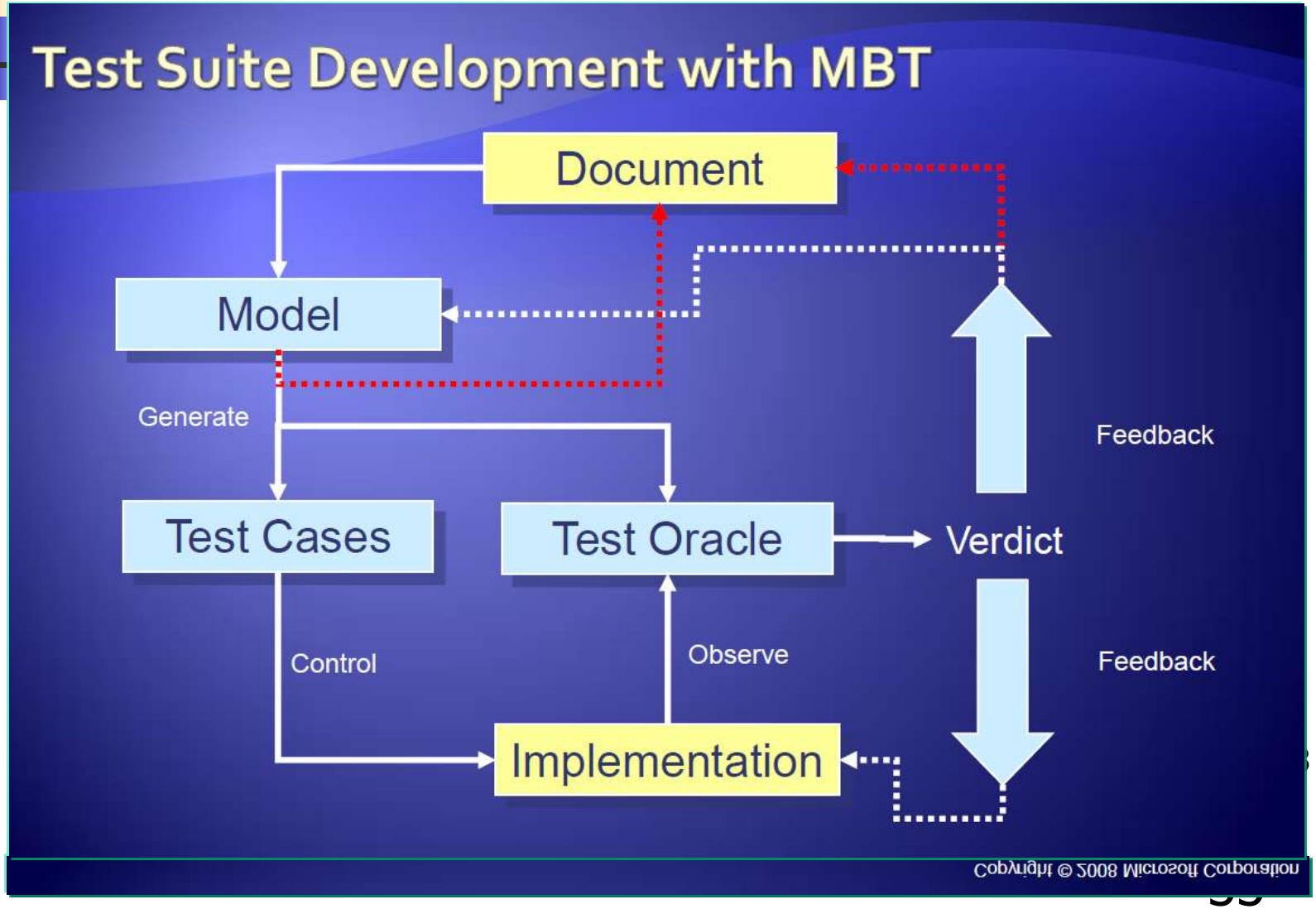
- **варианты постусловия**
- **самый простой**
-
- Contract.Ensures(Contract.Result<**double**>()*Contract.Result<**double**>() == x);
-
- **посложнее (с учетом неточного равенства)**
-
- Contract.Ensures(x != 0 || Contract.Result<**double**>() == 0);
- Contract.Ensures(x == 0 ||
  Math.Abs((Contract.Result<**double**>()*Contract.Result<**double**>() – x)/x)
  < 1.0e-15);
-
- **еще сложнее (с учетом неточного равенства и отрицательных аргументов)**
-
- Contract.Ensures(x >= 0 || Double.IsNan(Contract.Result<**double**>()));
- Contract.Ensures(x != 0 || Contract.Result<**double**>() == 0);
- Contract.Ensures(x <= 0 ||
  Math.Abs((Contract.Result<**double**>()*Contract.Result<**double**>() – x)/x)
  < 1.0e-15);

Ретроспектива развития языков программирования со средствами спецификации

- Gypsy 1975, Alphard 1976, Euclid 1977, **CLU 1975-1988**
- Anna 1990, ADL 1993, **Eiffel 1988,** SPARK 2003
- **JML 1999**, UniTESK 2000
- Spec# 2003
- Автоматическое доказательство теорем для поиска ошибок: ESC/Java 2002, Modula-3 1998

MS Interoperability Initiative



32

MS Interoperability Initiative (2)