

DIM0436

15. Programação e contrato

Richard Bonichon

20140923

Sumário

- 1 Programação por asserções
- 2 Programação por contrato
- 3 ACSL & E-ACSL

- 1 Programação por asserções
- 2 Programação por contrato
- 3 ACSL & E-ACSL

O que é uma asserção

Definição (Asserção)

- Uma asserção é um **predicado** num programa indicando o que o programador acha ser sempre verdadeiro nesse lugar.
- Uma asserção **falsa** durante uma execução do programa causa uma **falha de asserção**, geralmente terminando o programa.

Exemplo

```
x = 1;  
assert (x > 0);  
x++;  
assert (x > 1);
```

Assertões vs. comentários

Assertões permitem verificação no momento da execução de hipóteses que geralmente aparecem só nos comentários mas:

- a execução do programa sabe nada dos comentários
- comentários são muitas vezes
 - ▶ antiquados
 - ▶ dessincronizados do código fonte
- instruções `assert`, executada, não pode virar antiquada, porque se for o caso, vão falhar e **necessitar** uma atualização.

Terminação imediata

Recomendação 32

Crash Early

A dead program normally does a lot less damage than a crippled one.

– The Pragmatic Programmer

Recomendação 33

Use Assertions to Prevent the Impossible

Assertions validate your assumptions. Use them to protect your code from an uncertain world.

– The Pragmatic Programmer

Programar com asserções

```
double sqrt(double x);

int get_int(void);

double dist(double x, double y)
{
    double s = x * x + y * y;
    assert(s >= 0.0);
    return sqrt(s);
}

int main(int c, char **v){
    dist(get_int(), get_int());
}
```

- 1 Programação por asserções
- 2 Programação por contrato
- 3 ACSL & E-ACSL

Recomendação 31

Design with Contracts

Use contracts to document and verify that code does no more and no less than it claims to do.

– The Pragmatic Programmer

Vantagens da programação por contrato

- Método sistemático de desenvolvimento de *bug-free software*
- Um quadro geral para
 - ▶ depurar
 - ▶ testar
 - ▶ garantir qualidade
- Método de documentação dos componentes do software

Especificação

- O problema para melhorar a confiabilidade do software é a definição precisa, para todo componente, do que ele deve implementar.
- Só especificar o comportamento não pode garantir a implementação dela por o modulo mas:
 - ▶ se não especificar o que o modulo deve fazer, tem pouca chance que ele faça-o. (*Law of excluded miracle*)
 - ▶ na prática, o impacto da presença da especificação é grande

Especificação e correção

A especificação não garante a correção da implementação mas pode usá-la para fases de:

- testes
- depuração

Contrato

Definição (Contrato)

- Um **contrato** é escrito para que o **fornecedor** desenvolva uma tarefa para um **cliente**
- Cada participante espera alguns benefícios e aceita em retorno algumas obrigações
- Um **contrato** detalha os benefícios e as obrigações dos dois participantes

Exemplo

Contrato entre cliente e companhia aérea

	Obrigações	Benefícios
Cliente	Estar no aeroporto . 1h antes da saída Pagar bilhete 2 malas de 32kg no máximo	Chegar em Lisboa
Fornecedor	Levar cliente para Lisboa	Não é preciso levar um passageiro - atrasado - com bagagens inapropriadas - que não pagou bilhete

Exemplo no contexto do software

Tarefa

Inserir um elemento (associado a uma chave do tipo `string`) num dicionário de capacidade limitada

Contrato

	Obrigações	Benefícios
Cliente	Ter a certeza que: <ul style="list-style-type: none">- o dicionário não é cheio- a chave não é a palavra vazia	Obter um dicionário atualizado onde o novo elemento é associado a chave indicada
Fornecedor	Registrar o elemento no dicionário, associado a chave indicada	Não é preciso fazer alguma coisa se: <ul style="list-style-type: none">- o dicionário é cheio- a chave é a palavra vazia

Elementos básicos de um contrato

Pré-condições requisitos que devem ser satisfeitos **antes** a execução da rotina.

Pós-condições requisitos que devem ser satisfeitos **após** a execução da rotina

Invariantes propriedades que devem ser satisfeitas em todos os pontos de execução da rotina.

Linguagens com suporte para programação por contrato

Interno

- Eiffel
- SparkADA
- Racket
- Clojure
- D

Externo

Java Contracts for Java, JML, iContract ...

C ACSL

Python Contracts for Python, PyDBC

JavaScript Cerny.js, jsContract, ...

Exemplo (Eiffel)

```
put (x: ELEMENT; key: STRING) is
  -- Insert x so that it will be retrievable through key.
  require
    count <= capacity
    not key.empty
  do
    ... Some insertion algorithm ...
  ensure
    has (x)
    item (key) = x
    count = old count + 1
  end
```

- 1 Programação por asserções
- 2 Programação por contrato
- 3 ACSL & E-ACSL**

ACSL

Definição

- ACSL = ANSI C Specification Language
 - ACSL é uma linguagem lógica para escrever anotações nos programas C
 - O conceito fundamental ACSL é o **contrato de função**
 - Inspirado por JML
-
- ACSL é uma linguagem inicialmente dedicada à prova dedutiva (estática)
 - E-ACSL é um subconjunto executável dela par verificação no tempo de execução

Fundamentos lógicos de ACSL

- Lógica clássica da primeira ordem
- Conectivos lógicos
- Igualdade
- Quantificadores

Tabela sintética

Negação	<code>!P</code>
Conjunção	<code>P && Q</code>
Disjunção	<code>P Q</code>
Implicação	<code>P ==> Q</code>
Equivalência	<code>P <==> Q</code>
Cadeia de relações	<code>x < y == z</code>
Quantificador universal	<code>\forall int x; P</code>
Quantificador existencial	<code>\exists int x; P</code>

Elementos sintáticos ACSL

Elementos básicos de contrato

Pré-condições `requires`

Pós-condições `ensures`

Invariantes `loop invariant`, `ensures`

Asserções `asserts`

- `assigns`
- `behavior`
- `assumes`
- `terminates`
- `variant`
- ...

requires

- `requires` define um elemento da **pré-condição** da função
- A definição de pré-condições deve acontecer antes dos outros predicados.
- Pré-condições são definidas ao nível da função, antes dela.

Exemplo

```
//@ requires n >= 0;  
int fact(int n) {  
    int m = 1;  
    if (n != 0) {  
        while (n > 1) {  
            m = m * n;  
            n--;  
        }  
    }  
    return m;  
}
```

ensures

- `ensures` define um elemento da **pós-condição** da função
- A definição de pós-condições deve ser feitas após as pré-condições
- Pós-condições são definidas ao nível da função, antes dela.

Exemplo

```
//@ ensures \result >= 0;  
int abs(int n) {  
    if (n <= 0) return (-n);  
    return n;  
}
```

Observação

- `\result` é uma palavra-chave ACSL para falar do resultado duma função

assert

- assert define uma asserção
- Asserções podem ocorrer antes de qualquer instrução do programa.

Exemplo

```
//@ requires n >= 0;
int fact(int n) {
    int m = 1;
    if (n != 0) {
        while (n > 1) {
            m = m * n;
            n--;
        }
        //@ assert n == 1;
    }
    //@ assert m >= 1;
    return m;
}
```


Benefício de ACSL

Exemplo

```
int fact(int n) {
  int m = 1;
  if (n != 0) {
    while (n > 1) {
      m = m * n;
      n--;
    }
    /*@ assert n = 1;
  }
  return m;
}
```

Mensagem

- [kernel] user error: Assignment operators not allowed in annotations.

Exemplo

```
#include <assert.h>
int fact(int n) {
  int m = 1;
  if (n != 0) {
    while (n > 1) {
      m = m * n;
      n--;
    }
    assert(n = 1);
  }
  return m;
}
```

Mensagem

- ...

behavior & assumes

- Às vezes, funções podem ser complicadas e ter vários caminhos, lógicos ou concretos
 - ▶ A palavra-chave `behavior` permite isolar alguns desses caminhos
- Os comportamentos podem
 - ▶ declarar pós-condições adicionais, com `ensures`
 - ▶ ter hipóteses mais precisas, com `assumes`

Exemplo behavior

```
/*@ requires n >= 0;
   ensures \result >= 1;
   behavior is_zero:
     assumes n == 0;
     ensures \result == 1;
   behavior is_non_zero:
     assumes n > 0;
     ensures \result >= 1;
   disjoint behaviors is_non_zero, is_zero;
   complete behaviors is_zero, is_non_zero;
*/
int fact(int n) {
  int m = 1;
  if (n != 0) {
    while (n > 1) { m = m * n; n--; }
  }
  return m;
}
```

terminates

```
/*@  
assigns \nothing;  
terminates c>0;  
*/  
void f (int c) { while(!c); return;}
```

assigns

- Uma cláusula `assigns` lista os elementos cujos valores podem ser reescritos pela função.
- O comportamento normal é considerar todos os elementos como reescrevíveis

```
/*@ requires n > 0;
   requires \valid(p + (0..n-1));
   assigns \nothing;
   ensures \forall int i; 0 <= i <= n-1 ==> \result >= p[i];
   ensures \exists int e; 0 <= e <= n-1 && \result == p[e];
*/
int max_seq(int* p, int n);
```

loop invariant

- Anotação específica aos laços
- Consequência da dificuldade do tratamento dos laços por análise estática
- Forma de asserção para laços, expressando propriedades sempre verdadeiras

Exemplo

```
/*@ requires n >= 0;  
   ensures \result >= 1;  
   ensures n == 0 || n == 1;  
*/  
int fact(int n) {  
    int m = 1;  
    if (n != 0) {  
        //@ loop invariant n >= 1;  
        while (n > 1) { m = m * n; n--; }  
    }  
    return m;  
}
```

Definição

- E-ACSL é um subconjunto **executável** de ACSL
- Anotações ACSL são compiladas em asserções verificadas no **tempo de execução**

Onde achar ?

<http://frama-c.com/eacsl.html>

Exemplo E-ACSL

```
double sqrt(double x);

int get_int(void);

double dist(double x, double y)
{
    double s = x * x + y * y;
    /*@ assert(s >= 0.0); */
    return sqrt(s);
}

int main(int c, char **v){
    dist(get_int(), get_int());
}
```


Código gerado

```
/*@ assigns |result;
   assigns |result |from x; */
extern double sqrt(double x);

/*@ assigns |result;
   assigns |result |from |nothing; */
extern int get_int(void);

double dist(double x, double y)
{
    double s;
    double tmp;
    s = x * x + y * y;
    /*@ assert s >= 0.0; */
    e_acsl_assert(s >= 0.0, (char *)"Assertion", (char *)"dist",
                  (char *)"s >= 0.0", 9);
    tmp = sqrt(s);
    return tmp;
}
```

Uso de contratos

Verificação no tempo da execução

- Afeita a execução do programa
 - ▶ diminuição da velocidade de execução
 - ▶ pode introduzir efeitos secundários `assert(x = 0)`
- Não precisa de mais ferramentas uma vez compilada
 - ▶ mas precisa (geralmente) de uma biblioteca dedicada



Verificação semântica

- Não afeita a execução do programa
- Necessita geralmente uma bibliotecas / programas auxiliaários
- Necessita ferramentas adicionais
 - ▶ Núcleo lógico de cálculo
 - ▶ Provedores

Resumo

- 1 Programação por asserções
- 2 Programação por contrato
- 3 ACSL & E-ACSL

Referências

-  Patrick Baudin, Pascal Cuoq, Jean-Christophe Filiâtre, Claude Marché, Benjamin Monate, Yannick Moy, and Virgile Prevosto, *Acsl: Ansi/iso c specification language. preliminary design, version 1.8*, 2013, http://www.frama-c.cea.fr/download/acsl_1.8.pdf.
-  Andrew Hunt and David Thomas, *The pragmatic programmer: From journeyman to master*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

Perguntas ?



<http://dimap.ufrn.br/~richard/dim0436>