Инструменты дедуктивной верификации программ

Лекция №3

Если ваш единственный инструмент – молоток, то каждая проблема становится похожей на гвоздь

А. Маслоу

Александр Сергеевич Камкин

kamkin@ispras.ru

Установка Frama-C / AstraVer

https://forge.ispras.ru/projects/astraver/wiki

- **sudo apt-get** install gcc m4 make git
- **wget** https://raw.githubusercontent.com/ocaml/opam/master/shell/install.sh
- **sh** install.sh --version 2.0.9 # Конкретная версия opam!
- **opam** init -c 4.14.1 --disable-sandboxing
- **eval** $(opam env)
- **opam** repo add ispras https://forge.ispras.ru/git/astraver.opam-repository.git
- **opam** update
- ~~**opam** install depext~~
- ~~**opam** depext astraver~~
- **opam** install astraver
- **opam** install alt-ergo altgr-ergo
- **sudo** apt-get install cvc4 # Рекомендуется установить CVC4!
- **why3** config --detect

ACSL (ANSI C Specification Language)

- **Внешние аннотации (global annotations)**
  - Контракты функций (**`ensures`**, **`assigns`**, **`requires`**)
  - Вспомогательные определения
    - Переменные
    - Функции, предикаты, аксиоматики
    - Леммы
- **Внутренние аннотации (local annotations)**
  - Утверждения (**`assert`**)
  - Аннотации циклов (**`loop invariant`** / **`variant`**, **`assigns`**)
  - Вспомогательный код (**`ghost`**)

Пример внешней аннотации

```
/*@ // предусловие функции
  @ requires a >= 0;
  @ requires b > 0;
  @ requires \valid(r);
  @ // модифицируемый блок памяти
  @ assigns *r;
  @ // постусловие функции
  @ ensures \let q = \result; a == q * b + *r;
  @ ensures 0 <= *r < b;
  @*/
int idiv(int a, int b, int *r);
```

Пример внутренней аннотации

```
int idiv(int a, int b, int *r) {
  int q = 0;
  int p = a;
  /*@ loop invariant (a == q * b + p) && (0 <= p <= a);
    @ loop assigns q, p;
    @*/
  while(p >= b) {
    q++;
    p -= b;
  }
  /*@ assert (a == q * b + p) && (0 <= p < b); */
  *r = p;
  return q;
}
```

Базовые элементы ACSL

- **Логические константы:** `\true`, `\false`
- **Логические связки:** `&&`, `||`, `!`, `==>`, `<==>`, `^^`
- **Побитовые аналоги:** `&` , `|` , `~`, `-->`, `<-->`, `^`
- **Кванторы существования и всеобщности:**

  `\exists` T_1 x_1, …, T_n x_n; B

  `\forall` T_1 x_1, …, T_n x_n; B

- **Конструкции связывания переменных и именования выражений**:

  `\let` x = e; E id: E

- **Операции модификации структур и массивов:**

  {s **for** .$field$ = e} {a **for** [i] = e}

Сравнения: синтаксический сахар

Запись $e_1 \lessgtr_1 e_2 \lessgtr_2 e_3 \lessgtr_3 \ldots \lessgtr_{n-1} e_n$

эквивалентна

$(e_1 \lessgtr_1 e_2)\ \&\&\ (e_2 \lessgtr_2 e_3)\ \&\&\ \ldots\ \&\&\ (e_{n\_1} \lessgtr_{n-1} e_n)$

`0 <= r < b` **вместо** `0 <= r && r < b`

Типы данных

- **Базовые типы данных**
  - Машинные типы
    - **`char`**, **`short`**, **`int`**, **`long`**, **`float`**, **`double`**
  - Математические типы
    - **`integer`**, **`real`**
- **Составные типы данных**
  - Структуры
    - `type point = struct { real x; real y; };`
  - Массивы
    - `type triangle = point[3];`

Спецификационные переменные

```
/*@ // объявление спецификационной
  @ // переменной
  @ logic integer count = 0;
  @*/

//@ // обновление значения переменной
//@ ghost count++;
```

Предикаты и функции: явное определение

```
/*@ predicate IsMultiple(integer a, integer x) =
  @    a % x == 0;
  @
  @ predicate IsPrime(integer a) =
  @    a >= 2 && !(\exists integer x, y;
  @       x > 1 && y > 1 && a == x * y);
  @
  @ logic integer Mul(integer x, integer y) =
  @    x * y
  @*/
```

Аксиоматические определения

```
/*@ axiomatic GCD {
  @   logic integer gcd(integer a, integer b);
  @   axiom gcd_equal:
  @     \forall integer x;
  @       (x > 0) ==> (gcd(x, x) == x);
  @   axiom gcd_comm:
  @     \forall integer x, y;
  @       (x > 0 && y > 0) ==> (gcd(x, y) == gcd(y, x));
  @   axiom gcd_add:
  @     \forall integer x, y;
  @       (x > 0 && y > 0) ==> (gcd(x + y, y) == gcd(x, y));
  @ }
  @*/
```

Контракты функций

```
/*@ // предусловие функции
  @ requires a >= 0;
  @ requires b > 0;
  @ requires \valid(r);
  @ // модифицируемый блок памяти
  @ assigns *r;
  @ // постусловие функции
  @ ensures \let q = \result; a == q * b + *r;
  @ ensures 0 <= *r < b;
  @*/
int idiv(int a, int b, int *r);
```

Память: **assigns** и \valid

- **assigns** \nothing;
- **assigns** x;
- **assigns** a[0..n-1];
- **assigns** *(a + (0..n-1));
- \valid(&x)
- \valid(a)
- \valid(a + (0..n-1))

Ветви функциональности (named behaviors)

```
/*@ requires φ;        // общее предусловие функции
  @ assigns  L;        // память, модифицируемая во всех ветвях
  @ ensures  ψ;        // общее постусловие функции
  @
  @ behavior b1:       // ветвь функциональности
  @   assumes  A1;     // условие активации ветви
  @   requires φ1;     // предусловие ветви
  @   assigns  L1;     // память, модифицируемая в ветви
  @   ensures  ψ1;     // постусловие ветви
  @
  @ behavior b2:
  @   assumes  A2;
  @   requires φ2;
  @   assigns  L2;
  @   ensures  ψ2;
  @*/
```

```
/*@ requires φ && (A1 ==> φ1) && (A2 ==> φ2);
  @ assigns  L, L1, L2;
  @ ensures  ψ && (\old(A1) ==> ψ1) && (\old(A2) ==> ψ2);
  @*/
```

Ветви функциональности: пример

```
/*@ requires n >= 0 && \valid(x + (0..n-1));
  @ assigns  \nothing;
  @ ensures  -1 <= \result <= n-1;
  @
  @ behavior success:
  @   ensures \result >= 0 ==> x[\result] == v;
  @
  @ behavior failure:
  @   assumes \forall integer i, integer j;
  @     0 <= i < j <= n-1 ==> x[i] <= x[j];
  @   ensures \result == -1 ==>
  @     \forall integer i; 0 <= i <= n-1 ==> x[i] != v;
  @*/
int bsearch(double x[], int n, double v);
```

Утверждения и инварианты циклов

```
int idiv(int a, int b, int *r) {
  int q = 0;
  int p = a;
  /*@ loop invariant (a == q * b + p) && (0 <= p <= a);
    @ loop assigns q, p;
    @*/
  while(p >= b) {
    q++;
    p -= b;
  }
  /*@ assert (a == q * b + p) && (0 <= p < b); */
  *r = p;
  return q;
}
```

Инварианты циклов

- условие φ истинно перед входом в цикл
  (для цикла $\mathbf{for}$ — после инициализации)
- истинность φ сохраняется после исполнения тела цикла:
  - для $\mathbf{while}(B)\ P$ истинность φ сохраняется после исполнения $B; P;$
  - для $\mathbf{for}(I; B;\ U)\ P$ — после исполнения $B; P; U;$
  - для $\mathbf{do}\ P\ \mathbf{while}(B)$ — после исполнения $P; B;$
- память вне областей L остается неизменной

Дополнительные клаузы

- **`complete behaviors`**
- **`disjoint behaviors`**
- **`loop variant`**

Что почитать

- J. Burghardt, J. Gerlach, T. Lapawczyk. *ACSL By Example*
- V. Prevosto. *ACSL Mini-Tutorial*
- *ACSL: ANSI/ISO C Specification Language*
- *Практическое введение*
  http://astraver.linuxtesting.org/manuals/acsl/