ПОЛИТЕХ
Санкт-Петербургский
политехнический университет
Петра Великого

ИКНТ

Верификация и анализ программ
Языки спецификаций

2020

КСПТ
КОМПЬЮТЕРНЫЕ СИСТЕМЫ И ПРОГРАММНЫЕ ТЕХНОЛОГИИ

Языки спецификаций

- Автономные языки/системы спецификаций
  - VDM
  - Z-нотация
  - RAISE
  - …
- Встроенные языки спецификаций
  - ACSL
  - VCC
  - JML
  - …

Автономные языки спецификаций. VDM

- Vienna Development Method
- Используется для спецификаций компьютерных систем
- Разработан IBM в 1970-ые годы
- Включает в свой состав языки спецификаций
  - VDM-SL
  - VDM++
- Стандарт ISO принят в 1996 году

Автономные языки спецификаций. Z-нотация

- Формальный язык спецификаций
- Используется для спецификации программных систем и вычислительных систем
- Разработан в 1977 году
- В 2002 году принят как стандарт ISO
- Нотация использует в том числе не ASCII символы (!)

Автономные языки спецификаций. RAISE

- Rigorous Approach to Industrial Software Engineering
- Подход к спецификации программного обеспечения
- Включает язык спецификаций RSL (Raise Specification Language)
- Разработан в 1992 году
- Позволяет задавать сложные спецификации промышленных программных систем

Спецификация поведения с помощью выражений темпоральной логики

- Использование LTL или CTL для спецификации элементарных поведений
- Представление общего поведения как композиции элементарных формул
- Генерация управляющей программы на основе системы темпоральных уравнений
- Простая верификация новых свойств

Встроенные языки спецификаций

- Достоинства
  - хранение спецификаций вместе с реализацией
  - Возможность контекстных спецификаций
  - Поддержка процесса параллельной разработки и специфицирования
  - Возможность верификации ПО
- Недостатки
  - Спецификации «размазаны» по программному коду

Язык ACSL

- ANSI/ISO C Specification Language (ACSL) - язык спецификаций программ на языке C
- Разработан в 2008 году, INRIA
- Текущая версия ACSL v.1.16 (2020 год)
- Предназначен для поведенческой спецификации C-программ
- Используется в задачах дедуктивной верификации и статического анализа

Язык ACSL

- Использует специальный формат комментариев
- Реализован в продукте Frama-C
- Не используется в динамическом анализе (в отличие от контрактов)

Возможности языка ACSL

- Описание контрактов функций
- Аннотации операторов
  - Утверждения (assertion)
  - Аннотации циклов
  - Контракты операторов
- Аннотации завершения
- Спецификация логики
- Зависимости по данным
- Инварианты данных
- Встроенные предикаты

ACSL. Контракты функций

- Спецификация предусловий
  - `//@ requires`
- Спецификация постусловий
  - `//@ ensures`
  - Специальный примитив `\result` для доступа к результату функции
  - Специальный примитив `\old(x)` для доступа к начальному значению переменной x.
- Спецификация влияния на окружение
  - `//@ assigns`

ACSL. Контракты функций

- В контрактах допускается задание множественных поведений
- Каждое поведение может именоваться
- У каждого именованного поведения имеется селектор (assumes)
- Управление полнотой множественных поведений
  - `//@complete behaviors`
  - `//@disjoint behaviors`

ACSL. Пример контракта функции max_element

```
/*@
requires IsValidRange(a, n);
assigns \nothing;

behavior empty:
   assumes n == 0;
   ensures \result == 0;

behavior not_empty:
  assumes 0 < n;
  ensures 0 <= \result < n;

  ensures \forall integer i; 0 <= i < n ==> a[i] <= a[\result];
  ensures \forall integer i; 0 <= i < \result ==> a[i] < a[\result];

complete behaviors;
disjoint behaviors;
*/
size_type max_element(const value_type* a, size_type n);
```

ACSL. Пример контракта функции swap

```
/*@
requires \valid(p);
requires \valid(q);

assigns *p;
assigns *q;

ensures *p == \old(*q);
ensures *q == \old(*p);
*/
void swap(value_type* p, value_type* q);
```

ACSL. Утверждения и контракты операторов

- Утверждения (assertion) – логические условия, которые должны выполняться в текущей позиции
  - `//@ assert <condition>`
  - Размещаются в теле функций
  - Аналог локального постусловия
- Контракты операторов
  - Позволяют масштабировать контракты на уровень операторов
  - Привязываются к именованному поведению

ACSL. Аннотации циклов

- Инварианты циклов
  - `//@ loop invariant C;`
    - Условие C выполняется до начала (первой итерации) цикла
    - Условие C выполняется внутри тела цикла
  - `//@ loop assigns L;`
    - Гарантирует неизменность внутри цикла переменных, не входящих в L
- Варианты циклов
  - Специфицируют переменную, которая
    - Имеет целочисленный тип
    - Уменьшается на каждой итерации цикла
    - Имеет неотрицательное начальное значение
    - `//@ loop variant x;`

ACSL. Пример инварианта цикла

```
/*@ requires n >= 0 && \valid(t +(0..n -1));
  @ assigns \nothing ;
  @ ensures -1 <= \result <= n -1;
  @ behavior success :
  @   ensures \result >= 0 ==> t[\result] == v;
  @ behavior failure:
  @   assumes t_is_sorted : \forall integer k1 , integer k2;
  @       0 <= k1 <= k2 <= n-1 ==> t[k1] <= t[k2];
  @   ensures \result == -1 ==>
  @       \forall integer k; 0 <= k < n ==> t[k] != v;
  @*/
int bsearch(double t[], int n, double v) {
  int l = 0, u = n -1;
  /*@ loop invariant 0 <= l && u <= n-1;
    @ for failure : loop invariant
    @   \forall integer k; 0 <= k < n && t[k] == v ==> l <= k <= u;
    @*/
  while (l <= u ) {
    int m = l + (u-l )/2;
    if (t[m] < v) l = m + 1;
    else if (t[m] > v) u = m - 1;
    else return m;
  }
  return -1;
}
```

ACSL. Аннотации завершения

- Используются для спецификации свойств, позволяющих осуществить вывод факта завершения/незавершения программы
- Для циклов:
  - Для домена целых чисел:
    - `//@ loop variant x;`
  - Для остальных доменов (R – фундированное отношение на домене)
    - `//@ loop variant x for R;`

ACSL. Аннотации завершения

- Для функций:
  - Для домена целых чисел:
    - `//@ decreases x;`
  - Для остальных доменов (R – фундированное отношение на домене)
    - `//@ decreases x for R;`
  - Для незавершаемых функций:
    - `//@ terminates x;`
      - `Пример: //@ terminates \true;`

ACSL. Спецификация логики

- Предикаты
- Логические функции
  - Обычные
  - Рекурсивные
- Леммы
- Аксиоматические определения

ACSL. Инварианты данных

- Предназначены для задания свойств данных, которые должны выполняться во время функционирования программы
- Два типа инвариантов:
  - Инварианты глобальных переменных (global invariant) – гарантируют свойства переменных
  - Инварианты типов (type invariant)– гарантируют свойства всех переменных этих типов
- Два вида строгости инвариантов:
  - Сильные (strong) – должны выполняться в каждый момент времени
  - Слабые (weak) – должны выполняться на входе и выходе функций (но необязательно внутри)

ACSL. Инварианты данных

```
int a;
//@ global invariant a_is_positive : a >= 0 ;

typedef double temperature;
/*@strong type invariant
 temp_in_celsius (temperature t) =  t >= -273.15 ;
@*/

struct S {
  int f;
};
//@ type invariant S_f_is_positive (struct S s) = s.f
>= 0 ;
```

ACSL. Встроенные предикаты

- `\valid(p)` – безопасная разадресация указателя p (p != null)
- `\initialized(x)` – переменная x – инициализирована
- `\specified(p)` – p указывает на инициализированное значение
- …

```
int *f() {
  int a;
  return &a;
}

int *g(){
  int *p = f();
  //@ assert \specified(p);
  return p+1;
}
```

ACSL. Применение

- Верификация программ с помощью модуля к системе Frama-C
- Использование ACSL как базы для внешних верификаторов и статических анализаторов:
  - Верификация: ИСИ РАН, Новосибирск
  - Верификация: AstroVer, Москва
  - Верификация/стат. анализ: СПбПУ

Система верификации VCC

- “A Verifier for Concurrent C”
- Разработана в Microsoft Research
- Сайт: http://vcc.codeplex.com
- Интерактивная демонстрация: http://rise4fun.com/vcc
- Включает собственный язык спецификаций
- Спецификации записываются как часть языка и обрабатываются собственным препроцессором

VCC. Язык спецификаций

- Контракты (пред- и постусловия) функций
- Инварианты циклов
- Утверждения (assertions)
- Инварианты типов
- Контракты блоков кода
- Условия завершения
- «Чистые» функции
- Интерпретация памяти
- …

VCC. Примеры спецификаций

```
#include <vcc.h>

int max(int a, int b)
  _(ensures \result == (a > b ? a : b))
{
  if (a > b) return a;
    return b;
}

#include <vcc.h>
#include <limits.h>
int succ(int i)
_(requires i<INT_MAX)
_(ensures \result == i+1)
{
    return i+1;
}
```

VCC. Примеры спецификаций

```
#include <vcc.h>

#define SIZE 10

typedef struct Set {
  unsigned int len;
  int data[SIZE];

  _(invariant len < SIZE)
  _(invariant \forall unsigned int i, j; i < j && j < len
==> data[i] < data[j])
} Set;
```