# Лекция 4. Введение в ACSL

# Цель лекции

Познакомиться с основами языка спецификации ACSL и с верификацией в среде Frama-C.

# Содержание

# Предикаты и функции

В Why3 есть `predicate` и `function`. Они означают не предикаты и функции в смысле нашей лекции 1, а означают предикатные и функциональные символы в смысле курса «математическая логика». Это удобный способ объявлять новые функции. Тем более, что работать с ними будут пруверы и солверы, для которых мат.логика естественна.

# Вспоминаем мат.логику

- константы
- функциональные символы
- термы
- предикатные символы
- формулы

# Вспоминаем мат.логику

- $\Gamma \models \Delta$
- формулы-аксиомы, формулы-цели
- модели (интерпретации) — множества всех означиваний символов (функц.символов и предикат.символов функциями), выполняющих аксиомы
- задача выполнимости
- задача общезначимости (нам нужна она!) – все модели аксиом

# Методы Флойда и мат.логика

Раньше каждому оператору блок-схемы сопоставлялись функции. Теперь – формулы. Условия верификации – это формулы. Полная корректность — когда из аксиом общезначимы все условия верификации.

# Пример аксиом

Определяем деление при помощи функциональных символов:

```
function div (a b: int): int
function mod (a b: int): int
axiom def: forall a b: int.
  a >= 0 /\ b > 0 ->
    a == b * (div a b) + (mod a b) /\
    0 <= (mod a b) < b
```

# Пример аксиом

Определяем деление при помощи предикатного символа:

```
predicate is_div (a b d m: int)
axiom def: forall a b d m: int.
  a >= 0 /\ b > 0 ->
    (a == b * d + m /\
    0 <= m < b <-> is_div a b d m)
```

# Содержание

# ACSL, Frama-C, AstraVer

- ACSL = ANSI/ISO Specification Language
- Frama-C — фронтенд статического анализа Си.
- AstraVer — плагин к Frama-C, который строит условия верификации
- запуск: `frama-c -av source1.c source2.c ...`

# ACSL

- Спецификация пишется в комментариях `/*@ ... */`.
- Для функций (предусловия, постусловия) — комментарий перед заголовком.
- Для циклов (индуктивные утверждения, оценочные функции) — комментарий перед циклом.
- Содержимое комментария — аннотации.
- Аннотация — ключевое слово + формула + точка с запятой.

# Спецификация функции

```
/*@ requires -128 <= x <= 127;
      // requires - предусловие
    ensures \result >= 0;
    ensures \result == x || \result == -x;
      // ensures - постусловие
      // несколько аннотаций - конъюнкция
      // \result
*/
int short_abs(int x);
```

# Спецификация цикла

Точка сечения – перед условием. Фундированное множество – натур.числа и сравнение «строго меньше».

```
/*@ loop invariant /* индуктив.утв. */ ;
    loop variant /* оценоч.функ. */ ;
*/
```

# Содержание

# Версии 1 и 2

Нужно написать функцию, которая выполняет некоторое действие `N` раз.

- `i_u_loop_1.c` — пример ensures
- `i_u_loop_2.c` — сначала доказываем завершаемость и отсутствие ошибок, не связанных с функциональными требованиями (`safety`), пример спецификации цикла

# Версии 3 и 4

- `i_u_loop_3.c` — AstraVer генерирует условия верификации для доказательства отсутствия арифметических переполнений и преобразований чисел, меняющих значение, смотрим сгенерированную теорию Why3: значения Си-переменных типа `int` и `unsigned` «конвертируются» в неограниченные числовые значения, использование стратегий Split и Inline.
- `i_u_loop_4.c` — исправляем реализацию, теперь все условия верификации доказываются

# Содержание

# Читаем массив

Нужно написать функцию, которая получает массив из чисел и 2 числа и возвращает положительное значение, ноль или отрицательное значение в зависимости от того, какое из чисел встречается больше в массиве.

```
int who_more(int *a, int n, int x, int y);
```

# Спецификация функции

- `\valid(p)` – истинно, если указатель `p` валидный (равен адресу корректно выделенного объекта памяти)
- `who_more_1.c` – спецификация функции, квантор всеобщности `\forall`, использование блока `axiomatic` для описания синтаксиса и семантики функционального символа `count`

# Верификация: инстанцирование аксиомы, assert, lemma

- `who_more_2.c` – спецификация цикла; некоторые условия верификации не доказываются, т.к. инструмент не может «догадаться», какие нужно применять аксиомы
- `who_more_3.c` – аксиомы должны появиться в посылках условий верификации, как их добавить в посылку? Расширить индуктивное утверждение или расширить предикат пути. Чтобы расширить предикат пути, применяем аннотацию `assert` внутри спецификационного комментария просто в коде. Одно из условий — важное свойство, оформили в виде `lemma`.