Лекция 7. WhyML, автоматизация построения условий верификации

Цель лекции

Познакомиться с языком WhyML и научиться использовать его для автоматизации дедуктивной верификации.

Содержание

Общая характеристика

- Инфраструктура Why3 умеет генерировать условия верификации (`why3 ide file`).
- *WhyML* – функциональный язык со средствами формальной спецификации и дедуктивной верификации.
- Обычно выполняет роль промежуточного языка между языком программирования и средствами верификации.

Типы данных

- Целые числа: `use import int.Int`
- Пустой тип `unit`
- Декартово произведение `type T = (T1,T2)`
- Структуры, поля которых можно делать изменяемыми – единственный способ реализовать переменные и операцию присваивания `type S = { mutable v: T }`
- Поле структуры – это еще и функция, поэтому его имя входит в глобальную область видимости
- Типы могут быть полиморфными `type array 't`

Операции и выражения

- Любое вычисление – это выражение
- `e1 ; e2` – последовательность вычислений выражений
- `s.v <- e` – присваивание в изменяемое поле `v` структуры `s`
- `()` – пустое вычисление (выражение типа `unit`)
- `(expr,expr)` – кортеж
- `{v = 3}` – новое значение типа структуры, у которого есть поле `v` (такой тип единственный)
- условие `if-then-else`, циклы `while`, `for`

Функции и локальные переменные

- Локальная переменная: `let var = expr in ...`
- Разбиение кортежа: `let (v1,v2) = expr in ...`
- Неанонимная функция: `let function (arguments) : returntype = expr`
- Можно указать только заголовок функции, тогда вместо `let` надо написать `val`. Заголовок заканчивается перед символом равно.
- Функция нужна только для доказательства? После `let` надо написать `ghost`.
- Это лемма-функция? После `let` надо написать `lemma`.
- Нужна рекурсия? Надо приписать после `let` слово `rec`.

Исключения

- Исключения прерывают вычисление
- Исключения используются для возврата из функции до ее завершения
- Надо описать тип исключения (конструкция exception)
- try-блок, with-обработчик, raise-генерация исключения

Запись спецификации

- Предусловие и постусловие пишется между заголовком функции и символом равно.
- Предусловие состоит из 1 или более конструкций `requires {...}`
- Постусловие состоит из 1 или более конструкций `ensures {...}`
- Возвращаемое значение записывается как `result`.
- Несколько requires означает их конъюнкцию. Аналогично с ensures.
- Ни одного requires означает истину. Аналогично с ensures.

Методы Флойда

- Точки сечения выбраны жестко: перед проверкой условий циклов
- Индуктивное утверждение записывается так: `invariant {...}` и пишется после слова `do`, если речь о цикле `while`
- Несколько `invariant` возможно, это означает их конъюнкцию
- Фундированно множество выбрано жестко: целые числа и сравнение меньше
- Оценочная функция цикла записывается так: `variant {...}` и пишется там же, где `invariant`.
- Оценочная функция рекурсии записывается так же, но после предусловия.

Содержание

Доказательство примера

- Перепишите блок-схему вычисления квадратного корня в виде функции на языке *WhyML*. Напишите спецификацию и всё необходимое для методов Флойда.
- Завершите доказательство полной корректности: укажите недостающие леммы, докажите леммы по индукции или через последовательность других лемм, используйте вместо лемм вызов ghost-функции.

assert

- Вместо написания отдельной ghost-функции с пустым телом для доказательства некоторого утверждения удобнее воспользоваться конструкцией assert и не писать эту функцию.
- Синтаксис такой: `assert { condition }`
- Это аналогично ghost-функции с пустым телом и постусловием `condition`. Предусловие собрано из всех условий, которые находятся перед assert.

ghost-переменные

- Если надо доказывать много утверждений про квантор существования, может быть проще завести переменную только для целей доказательства и нужным образом ее инициализировать или даже изменять прямо в функции. Такие переменные называются ghost-переменными.
- Синтаксис такой: `let ghost var = ... in ...`