Лекция 4. Введение в ACSL

Цель лекции

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

Содержание

Отличия

В этой лекции будем рассматривать только работу с числами, массивы, структуры, указатели рассмотрим позднее.
Какие есть отличия блок-схем от функций в программе на языке Си? Какие есть общие моменты?

Мой вариант ответа

- в Си типы ограничены, операции могут приводить к переполнению, знаковое переполнение — это ситуация неопределенного поведения.
- в Си есть деление, а в блок-схемах его не было, т.к. оно было не определено при делении на 0.
- в Си есть локальные и глобальные переменные.
- в Си переменная - это область памяти, а в блок-схеме - нет областей памяти.
- в Си есть возможность компилятору выбрать порядок выполнения операций внутри выражения.

Компактнее

- в Си есть неопределенное поведение: операции корректно определены только на части значений их операндов.
- в Си есть глобальные переменные и области видимости.
- в Си программа может описывать разные порядки выполнения операций.

Как это реализовать?

Чтобы понять, как эти особенности выразить в блок-схемах, обсудим устройство системы дедуктивной верификации.

Содержание

Что нужно для применения методов Флойда

- записать блок-схему программы
- записать предусловие и постусловие
- указать точки сечения
- указать фунд.мн-во, инд.утв-я, оц.ф-и
- сгенерировать различные условия
- определить их истинность

Frama-C + AstraVer + Why3

- язык спецификации - ACSL - ANSI C Specification Language
- форма записи спецификации - спецификационные комментарии (аннотации в коде)
- точки сечения, фунд.мн-во выбрано за вас
- Вы пишете инд.утв-я и оц.ф-и (тоже на ACSL в спецификационных комментариях)

А автоматически

- Frama-C парсит код на Си и строит промежуточное представление
- Frama-C парсит спецификационные комментарии ACSL и строит предусловие и постусловие
- AstraVer по промежуточному представлению строит блок-схему и применяет методы Флойда
- Why3 вызывает солверы и пруверы
- солверы и пруверы доказывают истинность условий.

Солверы и пруверы

Солвер решает задачу решения уравнения: дана логическая формула со свободными переменными. Надо найти значения переменным, при которых формула истинна. Если получится определить, что уравнение не имеет решения, это будет здорово.

Прувер решает задачу поиска логического вывода: дана логическая система (набор констант, функциональных символов, предикатных символов, аксиом и правил вывода) и цель (формула), нужно найти вывод формулы, т.е. способ ее получить их аксиом, применяя лишь указанные правила вывода.

Использование солверов

Солверы - полностью автоматические. Пруверы - зачастую не автоматические.
Как использовать солвер в методах Флойда? Взять отрицание и сколемизировать - получится уравнение. Если солвер определит, что уравнение не имеет решения, значит исходное условие истинно. Нам это и нужно.
Солвер ничего не знает про язык Си, про вашу конкретно программу, какие там нужны математические функции - ему сначала надо дать определения всех нужных функций. Как это сделать?

Опять аксиомы

Решили использовать тот же подход с аксиомами. Правила вывода - логика первого порядка.
Какой вывод: у нас условие верификации было формулой с функциями, а для солвера нужна формула без функций, а с функциональными и предикатными символами.
Функциональному символу соответствует целый набор функций, предикатному символу - целый набор предикатов.
Все в целом эти функции должны удовлетворять аксиомам.
Это называется «интерпретация символа».

Пример

Определите арифметическое деление при помощи аппарата логики первого порядка.
Как определите деление на ноль? Никак, просто не будет соответствующей аксиомы. Это будет означать, что допускается любая функция, которая имеет любое значение при делении на 0. И при доказательстве истинности условий это нужно учитывать. Приведите сами примеры и убедитесь в этом.

Синтаксис axiomatic

А теперь то же на ACSL....

Содержание

Неопределенное поведение

Неопределенного поведения не должно быть в хорошей программе.
Можно применять методы Флойда, чтобы доказать отсутствие неопределенного поведения в программе? Да!
Как? Надо доказать, что в каждом месте программы, где возможно неопределенное поведение, соответствующие переменные не имеют значений, приводящих к неопределенному поведению.

safety

Такие условия генерируются и помещаются в раздел safety.
Туда же в safety помещаются и условия для доказательства завершимости.
Условия про то, что функция выполняет функциональные требования (т.е. условия верификации), помещаются в раздел behaviors.

Содержание

Сколько типов?

Для каждого типа данных нужны все свои арифметические операции со своими аксиомами. Что лучше: иметь много типов или один тип неограниченных чисел?
Конечно, неограниченный тип. Он называется integer. Все целочисленные переменные имеют этот тип и аксиому с границами значений. Отсутствие арифметических переполнений выражается в виде условий safety.