Лекция 11. Фрейм функции. Component-as-array. Более простая аксиоматизация модели памяти.

Цель лекции

Обсудить ряд важных дополнительных вопросов про аксиоматизацию блочной модели памяти.

Содержание

Мотивация

- Вы уже написали функцию, заменяющую одно значение на другое значение в массиве.
- Напишите еще одну функцию, чтобы проверить спецификацию первой функции. Функция принимает на вход массив (указатель на начало и размер) и 3 значения `v0`, `v1`, `v2`. Размер массива не должен быть меньше 2. Постусловие функции – что 0-й элемент массива становится равным `v0`, а 1-й элемент массива становится равным `v2`. Функция сначала присваивает в 0-й элемент массива значение `v0`, в 1-й элемент массива значение `v1`, затем вызывает первую функцию для части массива, с 1-го элемента до его конца.
- Почему одно из условий постусловия не доказывается?

Фрейм функции

- В спецификации первой функции не сказано, что может делать функция за рамками переданного массива. Значит, она может делать что угодно.
- Необходимо думать о границах области памяти (рамке, фрейме, footprint), которую разрешено изменять функции.
- Добавьте соответствующее постусловие к первой функции.
- Проверьте, что если второй функции передать еще один массив и с ним вторая функция ничего не делает, то этот факт доказывается!

Содержание

Цель

- Как моделировать Си-структуры? Массивы Си-структур?
- Первая идея – использовать WhyML-структуры один-в-один, те же поля, что и у Си-структур
- В модели памяти блоки будут состоять из указателей на структуру целиком
- Можно улучшить этот принцип моделирования

Component-as-array

- Отдельные поля одной структуры не могут находиться в одной области памяти (это не union).
- Создадим отдельный `memory map` для каждого поля структуры.
- (+) Упрощение верификации
- (-) Усложнение генерации модели программы (нужна автоматизация)

Содержание

Убираем тип block

- Тип `block` отличается от других тем, что его значения не меняются. Можно попробовать убрать этот тип из аксиоматизации.
- Типы `pointer` и `alloc_table` становятся типами-символами, т.к. в их определении использовался `block`. Тип `memory` не меняется.
- `alloc_table` был отображением блоков в их «размеры». Теперь вместо блоков у нас указатели. И каждому указателю надо сопоставить тот же «размер блока» и смещение. То есть размер блока и смещение зависят от `alloc_table` и `pointer`.
- Вместо размера блока и смещения можно использовать два других числа, взаимно однозначно им соответствующих: смещение от указателя к началу блока (`offset_min`) и к концу блока (`offset_max`).

Символы

Часть бывших не-символов становятся символами и наоборот:

```
type pointer 't
type alloc_table 't
function offset_min (a: alloc_table 't)
                    (p: pointer 't): int
function offset_max (a: alloc_table 't)
                    (p: pointer 't): int
function shiftP (p: pointer 't)(n: int): pointer 't
function subP(p p2: pointer 't): int
predicate same_block (p p2: pointer 't)
predicate valid (a: alloc_table 't)(p: pointer 't)
  = offset_min a p <= 0 <= offset_max a p
```

Остальное

Целиком аксиоматизацию можно посмотреть в библиотеке инструмента *AstraVer* примерно тут:
`.opam/4.07.1/lib/astraver/why3/core.mlw`