Лекция 12. Инструмент AstraVer: модель памяти, статическое разрешение синонимии указателей

Цель лекции

Обсудить некоторые проблемы дедуктивной верификации Си-программ и их решения, реализованные в инструменте AstraVer

Содержание

Убран 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`

Изучите сгенерированную модель программы

Запустите AstraVer на этом примере и изучите модель программы:

```
//@ requires \valid(v) && \valid(i);
void at(int *v, int n, int *i) {
    if (i - v > 2) {
        *v = *++i;
    }
}
```

Содержание

Формулировка проблемы

- Как моделировать Си-структуры? Массивы Си-структур?
- В WhyML есть структуры, но их нет в солверах
- Значит, тип структуры будет типом-символом, поля будут функциональными символами
- Но при изменении внутреннего элемента массива / поля получаются очень объемные условия верификации (проверьте!)

Component-as-array

- Заметим, что отдельные поля одной структуры не могут находиться в одной области памяти (это не union).
- Решение проблемы: отдельный `memory map` для каждого поля структуры.
- (+) Упрощение верификации
- (-) Усложнение генерации модели программы (нужна автоматизация)

Изучите сгенерированную модель программы

Запустите AstraVer на этом примере и изучите, как он разделил структуру по полям:

```
typedef struct {
    int capacity, size, *data;
} Vector;
//@ requires \valid(v) && \valid(v->data);
void at(Vector *v, int i) {
    if (v->size < v->capacity &&
        0 <= i < v->size) {
        v->data[i] = 0;
        ++ v->size;
    }
}
```

Содержание

Синонимия указателей

- Два указателя являются синонимами (алиасами), если изменение памяти по одному указателю влияет на память по другому указателю
- Проблема разрешения синонимии – определение того, какие указатели в тексте программы не являются синонимами.
- Это позволяет генерировать более эффективный код компилятору (запоминать считанные значения и не считывать их повторно)
- Известная исследовательская проблема в компиляторных технологиях

Синонимия указателей в дедуктивной верификации

- Важно разрешать синонимию и для дедуктивной верификации
- Если указатели не являются синонимами, то можно использовать для них разные переменные для модели памяти: «изменение» одной переменной не повлияет на другую переменную
- То есть это повышает эффективность разрешения условий верификации
- Постановка задачи: сопоставить каждому разыменованию каждого указателя некоторую переменную - модель памяти, чтобы у заведома несинонимов были разные модели памяти

Регионы

- Разделяем все указатели на классы эквивалентности (*регионы*), в одном классе находятся указатели, которые могут совпадать
- Изменение памяти по указателю из одного региона не изменяет память по указателям всех остальных регионов
- Поэтому для региона делаем отдельные переменные `alloc_table` и `memory`

Один класс эквивалентности

Когда два указателя должны находиться в одном регионе:

- когда они сравниваются в коде *даже на неравенство*
- когда они сравниваются в спецификации
- когда какой-то указатель из региона одного указателя сравнивается с каким-то указателем из региона второго указателя
- когда один получен из другого прибавлением числа, *каким бы оно ни было* (иначе алгоритм расстановки регионов по указателям не будет статическим)

Пример

Расставьте регионы в следующем примере кода:

```
int *a, *q, *r; ...
int *p = a;
while (p < q - 1) {
    if (*p == *r) ++*p;
    ++p;
}
```

Регионы для функций

- Нужны дополнительные неявные параметры для региона
- Для каждого явного параметра-указателя нужен параметр-регион, но некоторые параметры-указатели могут использовать один и тот же параметр-регион
- Связь между использованием параметра-указателя и использованием параметра-региона делается статически
- Если есть вызов этой функции, где у двух параметров-указателей один регион, то них будет общий параметр-регион
- Иначе у параметров-указателей будут разные параметры-регионы

Пример – отдельные регионы

В этом примере у a и b будут свои отдельные параметры-регионы

```
void f(int *a, int *b)
{
    *a = 0;
    *b = 0;
}
void g()
{
    int m[10];
    int n;
    f(&m[2], &n);
}
```

Пример – совместный регион

В этом примере у a и b будет один параметр-регион

```
void f(int *a, int *b)
{
    *a = 0;
    *b = 0;
}
void g()
{
    int m[10];
    int n;
    f(&m[2], &m[3]);
}
```

Упражнение

Почему не доказывается assert и что надо сделать для его доказательства?

```
/*@ requires \offset_min(s) == 0;
    requires \offset_max(s) == 0;
    ensures \base_addr(s) != \base_addr(\result);
*/
void *f(void *s)
{
    void *u = malloc(10);
    return u;
}
```

Упражнение

Почему возникает ошибка при построении модели тут? Исправьте ее (метка памяти – не то же, что регион!).

```
/*@
    axiomatic getValue {
        logic int *value(int *ar, integer);
    }

    predicate test1(int *ar, integer i) =
        *value(ar, i) == 0;
    predicate test2(int *ar, integer i) =
        test1(ar, i);
*/
```

Содержание

Решение в ACSL

- В ACSL есть предикат `\separated` от указателей. Он означает, что эти указатели не указывают на одни и те же области памяти (их разыменования независимы).
- Ответьте, почему этот предикат в общем случае не выразим в модели памяти AstraVer

Решение в AstraVer

- `\base_addr(p1) != \base_addr(p2)` – указатели не принадлежат одному блоку
- `p1 == p2 || p1 != p2` – разместить указатели в одном регионе
- Разделение по регионам вместо предиката `\separated`

Модульная верификация и разделение по регионам

- Вычисление регионов (т.е. переменные для модели памяти) требуют наличия всего исходного кода. А если есть только заголовочный файл?
- Для каждого файла-реализации для заголовочного файла придется верификацию всей программы делать сначала!
- AstraVer умеет делать некоторые предположения о регионах, исходя из заголовка (например, (не)константность указателя)