Лекция 10. Моделирование памяти Си-программы

Цель лекции

Построить аксиоматику для памяти Си-программы.

Содержание

Указатели в языке Си

- Указатель содержит адрес некоторого «объекта». Операция разыменования (прочитать или заменить «объект»).
- Разыменование невыравненного указателя – неопределенное поведение (не может проконтролировать компилятор).
- Разыменование невалидного указателя – неопределенное поведение (не может проконтролировать компилятор). Валидный указатель – тот, который можно корректно разыменовать.

Классы памяти

- Статическая память – распределяется до старта программы, существует все время работы (глобальные переменные всегда расположены в статической памяти, локальные переменные расположены в статической памяти, только если они объявлены со словом static)
- Стековая память – распределяется при вызове функции, освобождается при возврате из функции
- Динамическая память – распределяется в функциях `malloc`, `calloc`, `realloc`, освобождается в функции `free`.

Адресная арифметика

- Прибавление числа к указателю
- Вычитание указателей, сравнение указателей – возможны только для указателей, указывающих на один блок памяти. Для остальных указателей – неопределенное поведение.
- Указатели привязаны к блокам памяти!
- Блоки памяти не сравнимы!
- Блоки бывают освобожденными (freed).
- Приведение указателя к другому типу возможно, но результат стандартом не определяется.

Содержание

Побайтовая модель памяти

- Память – это массив байтов, указатель – это индекс в массиве и размер типа объекта указателя.
- Как промоделировать освобождение блока?
- Как проконтролировать выход за границы одного блока, если за ним идет другой блок?
- Нужны дополнения в эту модель! Слишком сложно для аксиоматизации.
- Хотя и можно точно промоделировать конкретную Си-платформу.

Блочная модель памяти

- Предположим, что не используется приведение типов указателей (и объединения), не сравниваются указатели из разных блоков памяти
- Указатель имеет блок и смещение от начала блока.
- Память – это 1) множество блоков, у каждого блока есть число: если оно неотрицательное, то это размер блока, если оно отрицательное, значит этот блок освобожден. 2) отображение из указателей в значения
- Указатели на объекты разных типов не лежат в одном блоке, поэтому для каждого типа объекта делаем свою память.

Оценка блочной модели памяти

- Более простая аксиоматика
- Не моделируется ограниченность памяти
- Не моделируется класс памяти
- Не моделируется calloc и realloc (нужно байтовое представление для заполнения)
- Не моделируется malloc с размером, не кратным размеру объекта указателя

Нормализация кода

Все остальное моделируется в такой модели памяти?

- Операция взятия адреса: да, ее можно устранить: если есть `x` и `&x`, то вместо `x` вводим новую переменную `xP`, которая хранит адрес `x`; вместо `x` используем `*xP`, вместо `&x` используем `P`.
- Переменные, на которые есть указатели: да, вместо них выделяем память при помощи `malloc`, освобождаем при помощи `free`.

Содержание

Вспомогательная теория

```
theory Map
  type map 'a 'b
  function get (m: map 'a 'b) (v: 'a): 'b
  function set (m: map 'a 'b) (v: 'a) (v2: 'b)
  axiom get_set: forall m: map 'a 'b,
      a a2: 'a, b: 'b.
      get (set m a b) a2 = (if a = a2 then b else
        get m a2)
end
```

Основные символы

```
type block
type pointer 't = (block, int)
type alloc_table 't = Map.map block int
type memory 't = Map.map (pointer 't) 't
```

Дополнительные символы

```
function blockOf (p: pointer 't): block
  = let (b, o) = p in b
function offsetOf (p: pointer 't): int
  = let (b, o) = p in o
function getSz (a: alloc_table 't) (b: block): int
  = Map.get a b
function setSz (a: alloc_table 't) (b: block)
 (sz: int): alloc_table 't = Map.set a b sz
function getV (m: memory 't) (p: pointer 't): 't
  = Map.get m p
function setV (m: memory 't) (p: pointer 't)
 (v: 't): memory 't = Map.set m p v
```

Дополнительные символы

```
function blockLen (a: alloc_table 't)
   (p: pointer 't): int = getSz a (blockOf p)
predicate valid (a: alloc_table 't) (p: pointer 't)
    =  0 <= offsetOf p < blockLen a p
predicate valid_range (a: alloc_table 't)
                      (p: pointer 't) (n: int) =
     0 <= offsetOf p < blockLen a p - n
predicate same_block (p p2: pointer 't) =
    blockOf p = blockOf p2
function shiftP (p: pointer 't)(n: int): pointer 't
 = let (b, o) = p in (b, o + n)
function subP (p p2: pointer 't): int
 = let (b, o) = p in let (b2, o2) = p2 in (b, o-o2)
```

Примитивы для операторов CALL

```
val readPtr (a: alloc_table 't) (m: memory 't)
            (p: pointer 't): 't
    requires { valid a p }
    ensures { result = getV m p }
val writePtr (a: alloc_table 't) (m: memory 't)
             (p: pointer 't) (v: 't): memory 't
    requires { valid a p }
    ensures { result = setV m p v }
val shiftPtr (p: pointer 't) (n: int): pointer 't
    ensures { same_block p result }
    ensures { offsetOf result = offsetOf p + n }
val subPtr (p: pointer 't) (p2: pointer 't): int
    requires { same_block p p2 }
    ensures { result = offsetOf p - offsetOf p2 }
```

Примитивы (2)

```
val malloc (m: memory 't): (memory 't, pointer 't)
    ensures { let (m2, p) = result in
        blockOf p was deallocated
        blockOf p was not used earlier (??????)
        blockOf p is allocated
        offsetOf p is 0
        sizes of other blocks are not changed
    }
```

Примитивы (3)

```
val free (m: memory 't) (p: pointer 't): memory 't
    requires { offset p is 0 }
    requires { blockOf p is allocated }
    ensures {
        blockOf p is deallocated
        sizes of other blocks are not changed }
```

Пример

Проверьте эффективность этой аксиоматизации модели памяти. Для этого решите следующие задания.

- Напишите на языке Си задачу прошлой лекции с массивом и заменой там одного значения на другое. Напишите модель программы. Напишите спецификацию. Докажите полную корректность модели программы относительно этой спецификации.

Вопросы

- Как промоделировать `null`?
- Как промоделировать Си-структуры? Массивы из Си-структур?
- Как промоделировать неявное приведение `void *` к любому указателю?