# Память и указатели

Как добавление еще одного уровня косвенности не *решает* проблемы, а **создает** их

---

Марат Ахин  Михаил Беляев

10 апреля 2018 г.

# Краткое содержание предыдущих серий

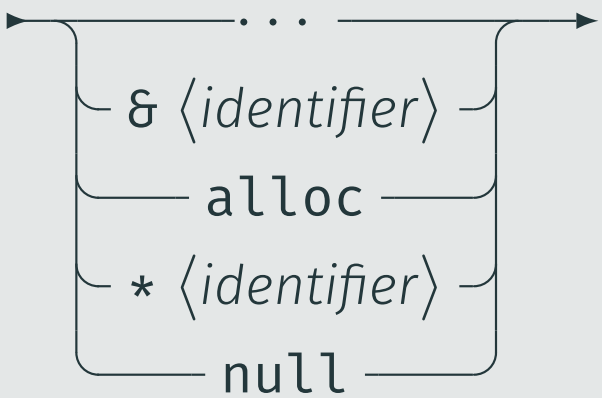
Статический анализ TIP

- Умеем выводить типы
- Умеем анализировать тело функции
    - Более того, умеем в потоково-чувствительный анализ!
- Умеем анализировать несколько функций

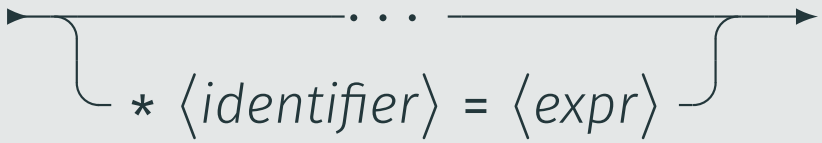
*А что мы не умеем, но что есть практически в любом языке программирования?*

# TIP: указатели

## ⟨*expr*⟩




## ⟨*stmt*⟩



# Почему указатели все портят?

```
var a,b,c,x;

a = 1;
b = 2;
*x = 42;
c = a + b; // wat?
```

# Почему указатели все портят?

```
var a,b,c,x;

a = 1;
b = 2;
*x = 42;
c = a + b; // wat?
```

- c = 3
- c = 43
- c = 44

*И это только самый простой случай...*

# Почему указатели все портят?

```
var a,b,c,foo;


a = 1;
b = 2;
(*foo)(&a, &b);
c = a + b; // waaat?
```

# Анализ указателей

- На что указывает тот или иной указатель?

или

- Указывают ли два указателя на одну и ту же память?

или

- Какую форму имеет доступная по указателям память?

# Анализ указателей

- На что указывает тот или иной указатель?
  - Анализ указателей (pointer analysis)

или

- Указывают ли два указателя на одну и ту же память?
  - Анализ псевдонимов (alias analysis)

или

- Какую форму имеет доступная по указателям память?
  - Анализ формы (shape analysis)

  *В большинстве случаев все эти проблемы являются эквивалентными*

# Анализ указателей

| Alias Mechanism | Intraprocedural May Alias | Intraprocedural Must Alias | Interprocedural May Alias | Interprocedural Must Alias |
|---|---|---|---|---|
| Reference Formals, No Pointers, No Structures | – | – | Polynomial[1, 5] | Polynomial[1, 5] |
| Single level pointers, No Reference Formals, No Structures | Polynomial | Polynomial | Polynomial | Polynomial |
| Single level pointers, Reference Formals, No Pointer Reference Formals, No Structures | – | – | Polynomial | Polynomial |
| Multiple level pointers, No Reference Formals, No Structures | $\mathcal{NP}$-hard | Complement is $\mathcal{NP}$-hard | $\mathcal{NP}$-hard | Complement is $\mathcal{NP}$-hard |
| Single level pointers, Pointer Reference Formals, No Structures | – | – | $\mathcal{NP}$-hard | Complement is $\mathcal{NP}$-hard |
| Single level pointers, Structures, No Reference Formals | $\mathcal{NP}$-hard[14] | Complement is $\mathcal{NP}$-hard | $\mathcal{NP}$-hard[14] | Complement is $\mathcal{NP}$-hard |

Table 1: Alias problem decomposition and classification

*Да, все в очередной раз очень плохо...*

# Представление памяти

*Для начала следует разобраться, а что такое память и как ее представлять?*

- Два типа памяти
  - Стек (статическая)
  - Куча (динамическая)

# Представление памяти

*Для начала следует разобраться, а что такое память и как ее представлять?*

- Два типа памяти
  - Стек (статическая)
  - Куча (динамическая)

## Стек

- Все относительно просто
  - Одна ячейка на каждую переменную программы
  - Что здесь не так?

# Динамическая память

## Куча

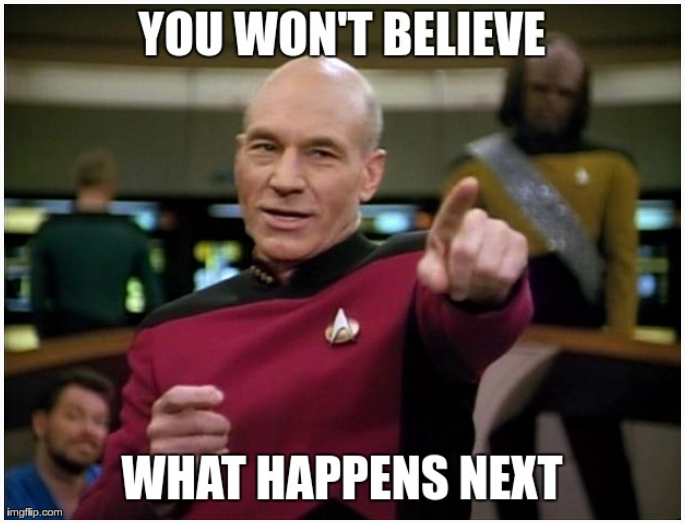
- Все сложно
    - Одна инструкция `alloc` может создать множество ячеек памяти
        - Циклы, рекурсия, вот это вот все
    - Как представить это множество?

# Динамическая память

- Одна ячейка на всю кучу
  - Очень просто
  - Очень неточно
- Одна ячейка на каждую *инструкцию* `alloc`
  - AKA allocation-site abstractrion
  - Что здесь не так?

# Чувствительность к контексту

- В нашем подходе все конкретные ячейки, выделенные в какой-то инструкции `alloc`, будут связаны с **одной** абстрактной ячейков
  - Даже между разными вызовами функций
- Можно попробовать сделать нашу модель контекстно-чувствительной



# Чувствительность к контексту

## Сигнатуры (строки) вызовов

- *Contexts* — это множество списков (строк) элементов множества $\{c_1, \ldots, c_N\}$ длины $\leq k$
- Для $k = 1$ сводится к предыдущему случаю
- По сути, мы моделируем стек вызовов на глубину $k$
- Меняя $k$, можно балансировать между размером множества контекстов и качеством анализа

*Можно применять все те же самые подходы,*
*что и для межпроцедурного анализа*
*Только контекст связывается с ячейками памяти*

# Массивы, структуры, связанные структуры данных

- Одна ячейка на всю структуру данных
  - Очень просто
  - Очень неточно
- Одна ячейка на каждый элемент
  - Как учитывать взаимосвязь между элементами?
- Несколько отдельных ячеек + ячейка «все остальное»
  - Попытка компромиса между производительностью и точностью

# Анализы, использующие анализ указателей

*Допустим, что мы решили все проблемы и у нас есть points-to отношение* $pt : Cells \longrightarrow 2^{Cells}$

- Как можно использовать эту информацию на практике?

**Разыменование нулевого указателя**

- Для каждой инструкции `*p` определить, может ли `p` быть нулевым указателем
  - Какая у нас решетка?
  - Как выглядят наши абстрактные вычисления?

# Решетка для анализа NPE


- Решетка *Null*
- `?` — в ячейке памяти *может* хранится `null`
- `NN` — в ячейке *точно не хранится* `null`

*Для анализа используем решетку Cells $\longrightarrow$ Null*

# Let's eval!

$$[\![x = alloc;]\!] = ???$$
$$[\![x = \&y;]\!] = ???$$
$$[\![x = y;]\!] = ???$$
$$[\![x = *y;]\!] = ???$$
$$[\![*x = y;]\!] = ???$$
$$[\![x = null;]\!] = ???$$
$$[\![n]\!] = JOIN(n) \quad \text{для любых других } n$$

$$JOIN(v) = \bigcup_{w \in pred(v)} [\![w]\!]$$

*Мы ничего не забыли?*

# Упрощение CFG

- `**x = *y + *z;`
  - Все операции с указателями упрощаются до следующих базовых операций
    - `x = *y;`
    - `*x = y;`
    - `x = &y;`
  - Много-много вспомогательных переменных
  - Сильно упрощает формулировку анализов

# Let's eval!

- Как посчитать `*x = y`?
  - `x` может указывать на множество ячеек памяти
  - Надо учесть это в наших абстрактных вычислениях
  - Это делается при помощи «слабой замены»

$$[\![\texttt{*x = y;}]\!] = store(JOIN(v), x, y)$$

$$store(\sigma, X, Y) = \sigma[\alpha \in pt(X) : \alpha \mapsto \sigma(\alpha) \sqcup \sigma(Y)]$$

# Let's eval!

- Как посчитать `x = *y`?
  - В нашей модели `x` указывает на одну ячейку памяти
  - Можно использовать «сильную замену»

$$[\![\text{x = *y;}]\!] = load(JOIN(v), x, y)$$

$$load(\sigma, X, Y) = \sigma[X \mapsto \sqcup_{\alpha \in pt(Y)} \sigma(\alpha)]$$

# Let's eval!

- Все остальные типы выражений также обновляют только одну ячейку памяти
- Для них тоже можно использовать «сильную замену»

$$
\begin{aligned}
[\![x = alloc;]\!] &= JOIN(v)[x \mapsto NN, alloc_v \mapsto ?] \\
[\![x = \&y;]\!] &= JOIN(v)[x \mapsto NN] \\
[\![x = y;]\!] &= JOIN(v)[x \mapsto JOIN(v)(y)] \\
[\![x = null;]\!] &= JOIN(v)[x \mapsto ?]
\end{aligned}
$$

# Замены revised

- Сильная замена: $\sigma[c \mapsto \mathit{newValue}]$
  - Возможна, когда $c$ указывает на одну ячейку
  - Работает для локальных переменных
- Слабая замена: $\sigma[c \mapsto \sigma(c) \sqcup \mathit{newValue}]$
  - Необходима, если $c$ может указывать на несколько ячеек памяти
  - Теряем точность, но с этим в принципе ничего не сделать

# Как можно повысить точность?

- Фильтрация по типам
  - Помечать ячейку памяти ее типом
  - Не учитывать все операции, не подходящие по типу
  - *Ломается, если есть небезопасные приведения типов*

# Как можно повысить точность?

- Фильтрация по типам
  - Помечать ячейку памяти ее типом
  - Не учитывать все операции, не подходящие по типу
  - *Ломается, если есть небезопасные приведения типов*
- Чувствительность к контексту
  - См. выше (и раньше)
  - Чем точнее мы учитываем контекст, тем сложнее становится анализ

# Как можно повысить точность?

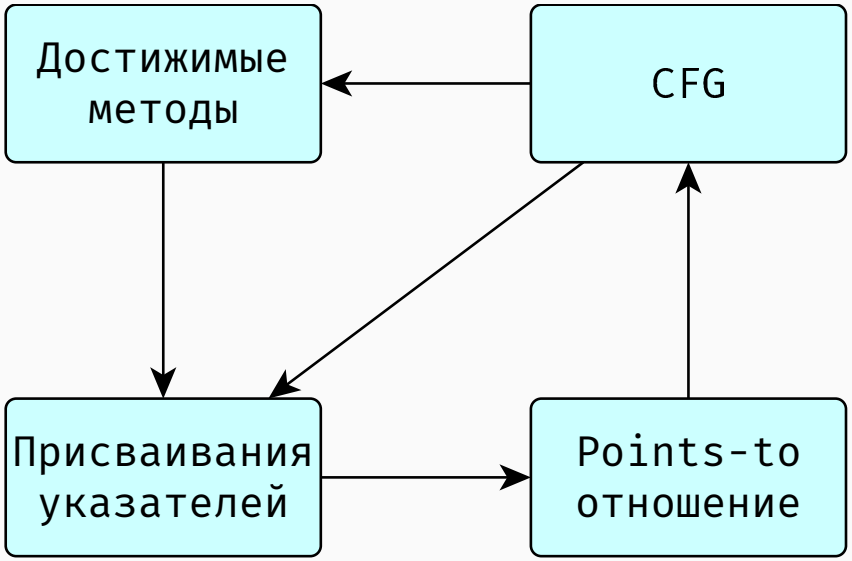
- Чувствительность к объектам
  - Актуально для языков с ООП
  - Контекст $\approx$ вызывающий объект
  - На практике работает *лучше* чувствительности к контексту

# Как можно повысить точность?

- Чувствительность к объектам
    - Актуально для языков с ООП
    - Контекст $\approx$ вызывающий объект
    - На практике работает *лучше* чувствительности к контексту
- Чувствительность к полям
    - AKA как учитывать структуры данных
    - Все варианты, рассмотренные раньше
    - И еще немного...

# Как можно повысить точность?

- Умное построение CFG




*С чего начать?..*

# Как можно повысить точность?

- Чувствительность к потоку управления
  - Большинство анализов указателей — потоково-нечувствительны
  - Их можно сделать потоково-чувствительными при помощи [не]больших изменений
  - Что при этом получается — в следующей серии

# Итого

- Отношение `point-to` помогает сделать другие анализы более точными
  - Или в принципе возможными =)
  - В зависимости от того, каким способом мы получаем отношение `points-to`, можно варьировать производительность и/или точность итогового статического анализа

*Осталась всего лишь одна проблема...*

# В следующей серии

Какие алгоритмы анализа указателей бывают и как они справляются с NP-полнотой?


akhin@kspt.icc.spbstu.ru

belyaev@kspt.icc.spbstu.ru