Анализ указателей и его друзья

Марат Ахин Михаил Беляев

17 апреля 2018 г.

Disclaimer

В предыдущей серии

- 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...

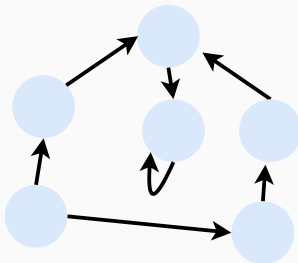
Disclaimer

Тут будет очень много повторов с предыдущей лекции

(Де)мотивирующий пример

```
foo(x,y) {
    val a,b,c;
    *x = 4;
    a = *x; // a = 4
    *y = 5;
    output a; // a = 4???
}
```

- Указатели *указывают* на области памяти
    - Будем называть их ячейками (memory cells)
    - Массивы и арифметику указателей пока забыли
- Нужно всегда помнить, что указатель может указывать **сам на себя**



Немного терминологии

Анализ псевдонимов (Alias analysis)

Для заданных указателей A и B, указывают ли они на одну и ту же ячейку памяти?

Анализ цели (Points-to analysis)

Для заданного указателя A, на какую ячейку памяти он указывает?

May-анализ указателей

Анализ псевдонимов (Alias analysis)

Для заданных указателей A и B, могут ли они указывать на одну и ту же ячейку памяти?

Анализ цели (Points-to analysis)

Для заданного указателя A найти набор ячеек, на которые он может указывать

Эти два анализа эквивалентны. На основе одного всегда можно построить другой, и наоборот

Анализ псевдонимов (Alias analysis)

Для заданных указателей A и B, могут ли они указывать на одну и ту же ячейку памяти?

Анализ цели (Points-to analysis)

Для заданного указателя A найти набор ячеек, на которые он может указывать

Альтернативные формулировки анализа указателей

Анализ множеств пседонимов (Alias set analysis)

- Разбивает все указатели на множества, которые могут указывать на одни и те же ячейки
- Строго говоря, более слабый анализ, чем анализ псевдонимов

Анализ формы (shape analysis)

- Какую структуру имеет достижимая по указателям область памяти?
- По сути, другое описание points-to analysis

Свойства алгоритмов анализа

- Как правило, реальные анализы нечувствительны к потоку
    - Чувствительная к потоку реализация анализа псевдонимов неразрешима
- Анализы могут быть *направленные* и *ненаправленные*
    - Ненаправленный анализ аналогичен анализу множеств псевдонимов

Дальнейшие допущения

- Считаем, что все обращения к памяти имеют один из 3 видов:

$$X = \&Y$$
$$*X = Y$$
$$X = *Y$$

- Что нам ещё нужно?

$$X = \texttt{alloc}$$
$$X = Y$$

- Все `alloc`'и пронумерованы (будем обозначать их как $alloc_N$)

Алгоритм Стенсгаарда

- Давайте остановимся на простом варианте (alias sets)
- В этом случае направление присваивания не имеет значения

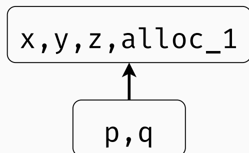
`X = alloc`	$[\![X]\!] = \&[\![alloc_1]\!]$
`X = Y`	$[\![X]\!] = [\![Y]\!]$
`X = &Y`	$[\![X]\!] = \&[\![Y]\!]$
`*X = Y`	$[\![X]\!] = \&\alpha \wedge [\![Y]\!] = \alpha$
`X = *Y`	$[\![Y]\!] = \&\alpha \wedge [\![X]\!] = \alpha$

Алгоритм Стенсгаарда: пример

```
var p,q,x,y,z;
p = alloc;
x = y;
x = z;
*p = z;
p = q;
q = &y;
x = *p;
p = &z;
```

$$[\![p]\!] = \&[\![alloc_1]\!]$$
$$[\![x]\!] = [\![y]\!]$$
$$[\![x]\!] = [\![z]\!]$$
$$[\![p]\!] = \&[\![z]\!]$$
$$[\![p]\!] = [\![q]\!]$$
$$[\![q]\!] = \&[\![y]\!]$$
$$[\![x]\!] = \&[\![p]\!]$$
$$[\![p]\!] = \&[\![z]\!]$$

```
var p,q,x,y,z;
p = alloc;
x = y;
x = z;
*p = z;
p = q;
q = &y;
x = *p;
p = &z;
```



$[\![p]\!] = \&[\![alloc_1]\!]$

$[\![x]\!] = [\![y]\!]$

$[\![x]\!] = [\![z]\!]$

$[\![p]\!] = \&[\![z]\!]$

$[\![p]\!] = [\![q]\!]$

$[\![q]\!] = \&[\![y]\!]$

$[\![x]\!] = \&[\![p]\!]$

$[\![p]\!] = \&[\![z]\!]$

Кто у нас умеет решать такого рода уравнения?

Алгоритм Стенсгаарда: солвер

Кто у нас умеет решать такого рода уравнения?

Давным давно, в далёкой-далёкой галактике...

Унификация

Есть набор равенств вида:

k(α,b,β) = k(f(β,γ),γ,d(γ))

Решение — набор значений переменных (**подстановка**), таких, что они делают эти равенства верными:

$$\alpha = f(d(b), b)$$

$$\beta = d(b)$$

$$\gamma = b$$

Решением этой задачи занимаются **унифицирующие солверы**

Алгоритм Стенсгаарда: солвер

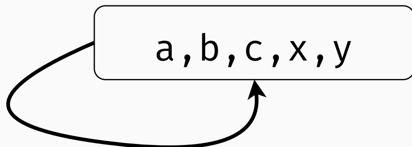
- Унифицирующий солвер возвращает равенства вида $X = Y = \&Z$
- Каждое такое выражение и есть alias set
- Как и в случае с типами, они могут быть рекурсивными

```
var a,b,c,x,y;
...
x = &a;
b = &y;
*y = c;
c = &y;
x = &y;
a = b;
```

Алгоритм Стенсгаарда: солвер

- Унифицирующий солвер возвращает равенства вида $X = Y = \&Z$
- Каждое такое выражение и есть alias set
- Как и в случае с типами, они могут быть рекурсивными

```
var a,b,c,x,y;
...
x = &a;
b = &y;
*y = c;
c = &y;
x = &y;
a = b;
```



Алгоритм Стенсгаарда

- Унификация на основе union-find имеет *почти линейную* сложность
- Это очень неплохо для анализа указателей

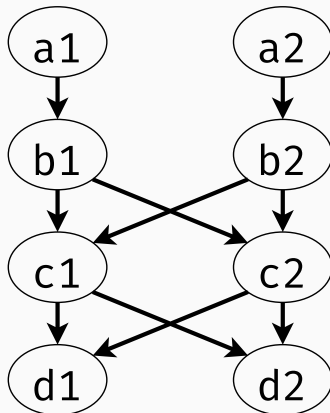
$$\mathit{pointsTo}(X) = \{c \in \mathit{Cells} \mid [\![\mathrm{X}]\!] = \&[\![\mathrm{c}]\!]\}$$

Алгоритм Стенсгаарда: проблемы

```
a1 = &b1;
b1 = &c1;
c1 = &d1;
a2 = &b2;
b2 = &c2;
c2 = &d2;
b1 = &c2;
```

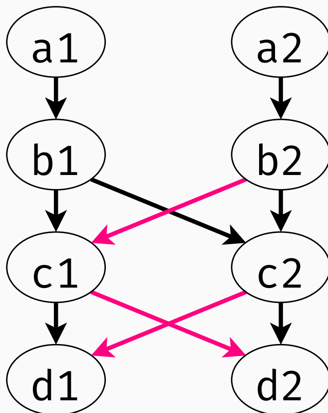
Алгоритм Стенсгаарда: проблемы

```
a1 = &b1;
b1 = &c1;
c1 = &d1;
a2 = &b2;
b2 = &c2;
c2 = &d2;
b1 = &c2;
```



Ничего не смущает?

```
a1 = &b1;
b1 = &c1;
c1 = &d1;
a2 = &b2;
b2 = &c2;
c2 = &d2;
b1 = &c2;
```



Интерлюдия: the cubic framework

Многие проблемы, сформулированные над множествами, можно свести к набору ограничений одного из видов:

- $t_i \in x_j$
- $t_i \in x_j \Rightarrow x_m \subseteq x_n$

где

- $t_1, t_2, \ldots, t_N$ — набор *токенов*
- $x_1, x_2, \ldots, x_K$ — набор *переменных*, соответствующих множествам токенов

Можно показать, что минимальное решение всегда существует

The cubic framework: солвер

- Структура решения представляет собой DAG, в вершинах которого хранится:
    - Множество токенов
    - Отображение из токенов в множества пар переменных
- Каждой переменной соотвествует нода DAG
- Уравнения обрабатываются по одному, структура всегда содержит минимальное решение

Для уравнений вида $t_i \in x_j$:

$node \Leftarrow nodes(x_j)$

$tokens(node) \Leftarrow tokens(node) \cup \{t_i\}$

if $node(t_i) \neq []$ **then**

  for all $\{x_k, x_n\} \in node(t_i)$ **do**

    add edge from $nodes(x_k)$ to $nodes(x_n)$

  end for

  $node(t_i) \Leftarrow \{\}$

end if

The cubic framework: алгоритм решения

Для уравнений вида $t_i \in x_j \Rightarrow x_m \subseteq x_n$:

$node \Leftarrow nodes(x_j)$

if $t_i \in tokens(node)$ **then**

add edge from $nodes(x_m)$ to $nodes(x_n)$

else

$node(t_i) \Leftarrow node(t_i) \cup \{\{x_m, x_n\}\}$

end if

The cubic framework: алгоритм решения

Если при добавлении дуги в граф получился цикл

- Соотвествующие вершины сливаются вместе
  - Сливаются множества токенов
  - Сливаются множества пар для каждого токена

The cubic framework: свойства

- В качестве реализации множества обычно используют Bitset
- Можно показать, что для множеств с константным добавлением общее время работы алгоритма не превышает $O(N^3)$
- Отсюда и название
- На практике ~ $O(N^2)$, что тоже не всегда приемлемо

- Общий случай *set constrains* имеет минимальную сложность $O(2^{2^N})$

Алгоритм Андерсена или зачем это было нужно

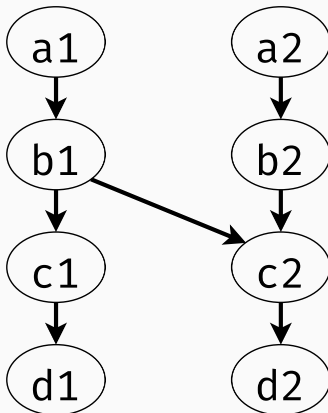
- Alias sets нас как-то не всегда устраивают
- Во многих случаях полученная аппроксимация слишком неточная
- Почему?
    - Потому что points-to множества слишком агрессивно объединяются
- Что делать?
    - Разрешить множествам частично пересекаться

Алгоритм Андерсена

`X = alloc`	$\&alloc_1 \in [\![X]\!]$
`X = Y`	$[\![Y]\!] \subseteq [\![X]\!]$
`X = &Y`	$\&Y \in [\![X]\!]$
`*X = Y`	$\&\alpha \in [\![X]\!] \Rightarrow [\![Y]\!] \subseteq [\![\alpha]\!]$
`X = *Y`	$\&\alpha \in [\![Y]\!] \Rightarrow [\![\alpha]\!] \subseteq [\![X]\!]$

NB: все уравнения удовлетворяют cubic framework

```
a1 = &b1;
b1 = &c1;
c1 = &d1;
a2 = &b2;
b2 = &c2;
c2 = &d2;
b1 = &c2;
```



Алгоритм Андерсена

- $O(N^3)$, как и ожидалось от cubic framework
- Значительно более точный, чем алгоритм Стенсгаарда
- Но дьявол кроется в деталях, такая точность не всегда нужна

$$pointsTo(X) = \{c \in Cells \mid \&[\![\mathrm{c}]\!] \in [\![\mathrm{X}]\!]\}$$

Context-sensitive alias analysis

- Что делать, если мы хотим межпроцедурность?
- Подходят стандартные подходы
  - Функциональная контекстная чувствительность
  - Моделирование стека на определённую глубину
- Будет ли и дальше работать cubic framework?
  - Вот и подумайте =)

Flow-sensitive alias analysis

- Что делать, если мы хотим потоковую чувствительность?
- Придётся возвращаться к монотонному фреймворку

$$States = 2^{Cells \times Cells}$$

По сути, это решётка points-to графов, описанная множеством дуг

X `= alloc`	$[\![v]\!] = JOIN(v) \downarrow X \cup \{(X, alloc_i)\}$
X_1 `=` &X_2	$[\![v]\!] = JOIN(v) \downarrow X_1 \cup \{(X_1, X_2)\}$
X_1 `=` X_2	$[\![v]\!] = assign(JOIN(v), X_1, X_2)$
X_1 `=` *X_2	$[\![v]\!] = load(JOIN(v), X_1, X_2)$
*X_1 `=` X_2	$[\![v]\!] = store(JOIN(v), X_1, X_2)$
X `= null`	$[\![v]\!] = JOIN(v) \downarrow X$

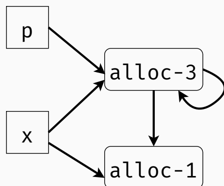
$$\sigma \downarrow X = \{(s, t) \in \sigma \mid s \neq X\}$$
$$JOIN(v) = \bigcup w \in pred(v) [\![w]\!]$$

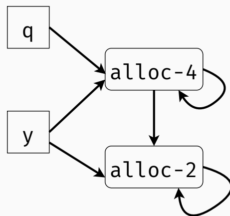
Flow-sensitive alias analysis

$$
\begin{aligned}
assign(\sigma, x, y) &= \sigma \downarrow x \cup \{(x, t) \mid (y, t) \in \sigma\} \\
load(\sigma, x, y) &= \sigma \downarrow x \cup \{(x, t) \mid (y, s) \in \sigma, (s, t) \in \sigma\} \\
store(\sigma, x, y) &= \sigma \cup \{(s, t) \mid (x, s) \in \sigma, (y, t) \in \sigma\}
\end{aligned}
$$

Flow-sensitive alias analysis: пример

```
var x,y,n,p,q;
x = alloc; y = alloc;
*x = null; *y = y;
n = input;
while (n>0) {
    p = alloc; q = alloc;
    *p = x; *q = y;
    x = p; y = q;
    n = n - 1;
}
```





Flow-sensitive alias analysis: анализ

$$pointsTo(X, v) = \{t \mid (X, t) \in [\![\mathrm{v}]\!]\}$$

- Точнее, чем анализ Андерсена
  - `x = &y; x = &z`
  - Андерсен: $pointsTo(x) = \{y, z\}$
  - FSAA: $pointsTo(x) = \{z\}$
- Но гораздо ресурсозатратнее
  - Вспомните формулу и посчитайте дома самостоятельно

Type-based alias analysis

- В языках со строгой системой типов указатели нельзя так просто превращать друг в друга
- Можно анализировать указатели на разные типы *отдельно*
    - Это совместимо с любым анализом, описанным в лекции
    - Значительно уменьшает пространство поиска
- В C это тоже частично верно, но по правилам strict aliasing

Как анализы указателей используются на практике

- В реальных системах анализа строится *стек анализов*
- Если верхний анализ выдал точный ответ — следующий не вызывается
- Как правило, используется решётка $\{NoAlias, MustAlias\}$

Сегодня мы многое поняли

- Если в языке есть указатели (**или ссылки**), от анализа указателей не убежать
- Практически любой анализ данных требует знаний от АА
- Анализы указателей настолько сильно отличаются по точности и производительности, что их выбор может стать критической точкой
- На практике, даже анализ Андерсена может оказаться слишком затратным

Что сегодня не рассмотрели:

- Сложные структуры данных (указатели со смещениями)
- Полный межпроцедурный анализ указателей

В следующей серии