Теория решеток или

как представлять состояние программы, чтобы потом не было ~~стыдно~~ сложно анализировать

Марат Ахин Михаил Беляев

27 февраля 2018 г.

Flow sensitivity

- Рассмотренный нами статический анализ (анализ типов) является потоково-*нечувствительным*
  - Типы выражений обычно не зависят от потока выполнения или потока данных программы[1]
  - Уравнения анализа строятся по исходному коду программы (AST)
- Большинство других видов анализа являются потоково-*чувствительными*
  - Результат зависит от порядка выполнения операций
  - Уравнения анализа строятся по графу потока управления (CFG)

[1] Кто вспомнил про Котлин, тот молодец

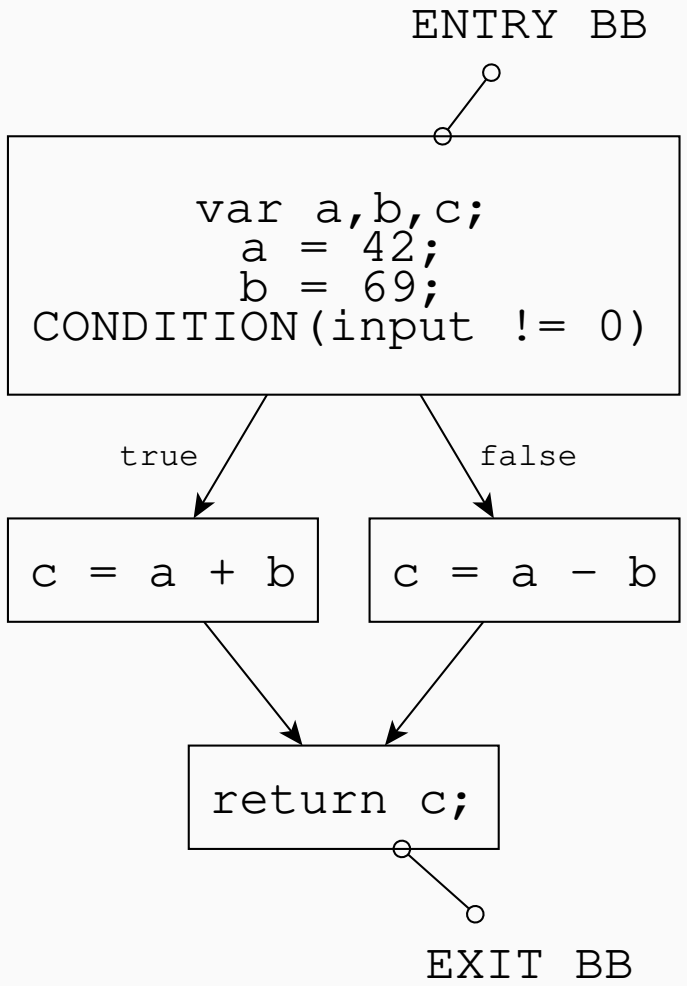
Control Flow Graph

Граф потока управления

- Представляет все возможные пути выполнения программы
- Состоит из *базовых блоков* и *переходов* между ними
  - Базовый блок — последовательность инструкций, всегда выполняющаяся последовательно
  - Для любого ребра $A \longrightarrow B$ верно, что $outdegree(A) > 1 \vee indegree(B) > 1$

Control Flow Graph

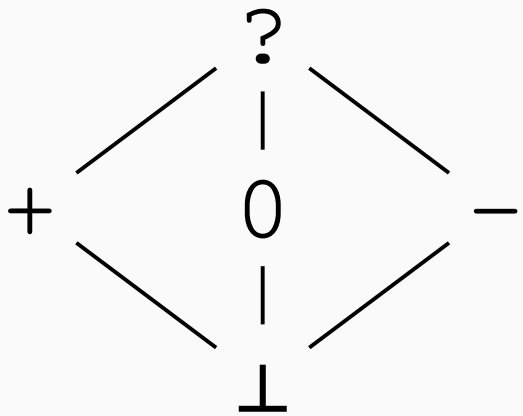
```
var a,b,c;
a = 42;
b = 69;
if (input) {
  c = a + b;
} else {
  c = a - b;
}
return c;
```



Анализ знака

Давайте попробуем определить статически знак выражения в TIP'е?

- Нам нужен какой-то *абстрактный домен*
  - Абстрактная информация для переменной программы
  - Может отличаться в разных точках программы



А почему нам не подходит конкретный домен?

Пример

```
var a,b;
a = 42;
b = a + input;
a = a - b;
```

$$x_1 = [a \mapsto ?, b \mapsto ?]$$

$$x_2 = x_1[a \mapsto +]$$

$$x_3 = x_2[b \mapsto x_2(a) + ?]$$

$$x_4 = x_3[a \mapsto x_3(a) - x_3(b)]$$

Правила вывода

Нас интересует $\sigma : [\![n]\!]$ — *абстрактное* состояние программы *после* вершины n из CFG

$$[\![\text{var x;}]\!] = JOIN(n)[x \mapsto ?]$$
$$[\![\text{x = E;}]\!] = JOIN(n)[x \mapsto eval(JOIN(n), E)]$$
$$[\![\text{smth;}]\!] = JOIN(n)$$

$JOIN(n)$ каким-то образом объединяет информацию из предков вершины n

Больше абстракции богу абстракции

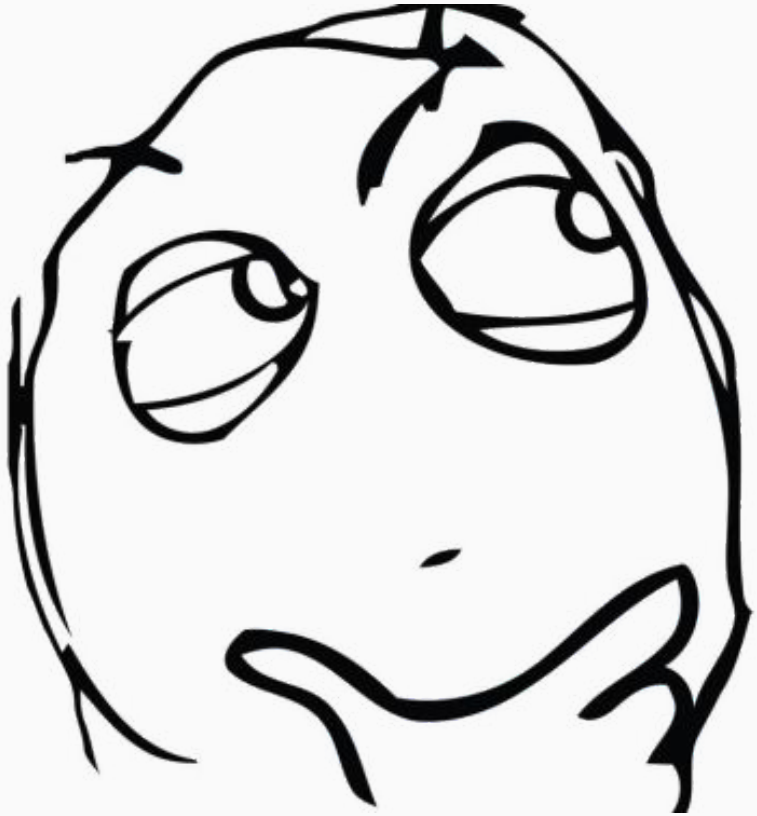
Что такое `eval`*?*

Чтобы работать в *абстрактном* домене, нужны *абстрактные* вычисления

$$
\begin{aligned}
eval(\sigma, INT) &= \overline{sign}(INT) \\
eval(\sigma, E_1 \ op \ E_2) &= eval(\sigma, E_1) \ \overline{op} \ eval(\sigma, E_2) \\
eval(\sigma, smth) &= \sigma(smth)
\end{aligned}
$$

$\overline{op}$ — *это* абстрактный *оператор, работающий в* абстрактном *домене*

Абстрактные операторы для TIP’а



Нехватка точности

$$eval\big(+\ /\ +\big) \ \mapsto\ ?$$

$$eval\big((42 > 0) == 1\big) \ \mapsto\ ?$$

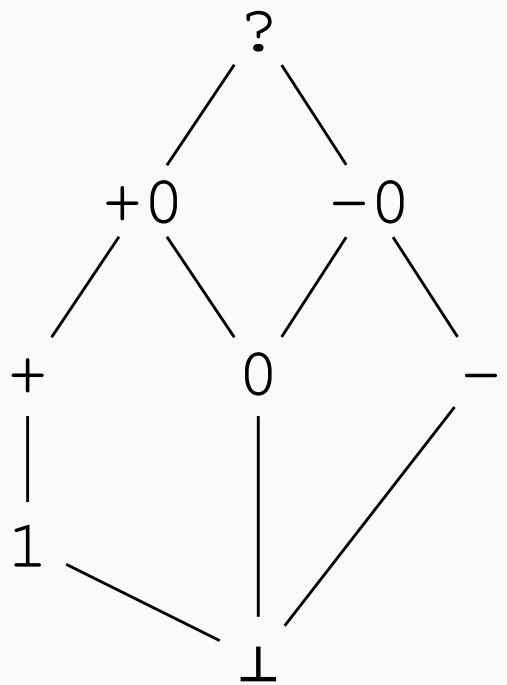
Что-то не хватает...

Нехватка точности

$$eval(+ \ / \ +) \ \mapsto \ ?$$

$$eval((42 > 0) == 1) \ \mapsto \ ?$$

Что-то не хватает...



Решетки

Почему наш уточненный абстрактный домен выглядит именно так?

Потому что для анализа очень здорово, если домен является *решеткой*

- Решетка — это частично упорядоченное множество (poset), для любых двух элементов которого есть точная верхняя и точная нижняя грани

Poset

- Poset — это множество S, для которого задано частичное отношение порядка $\sqsubseteq$, которое является
  - рефлексивным ($\forall x \in S : x \sqsubseteq x$)
  - транзитивным ($\forall x, y, z \in S : x \sqsubseteq y \wedge y \sqsubseteq z \Rightarrow x \sqsubseteq z$)
  - анти-симметричным ($\forall x, y \in S : x \sqsubseteq y \wedge y \sqsubseteq x \Rightarrow x = y$)

Грани

Пусть $X \subseteq S$

- $y \in S$ — верхняя грань X (обозначается как $X \sqsubseteq y$), если $\forall x \in X : x \sqsubseteq y$
- Точная верхняя грань $\sqcup X$ — минимальная из всех верхних граней
  - $X \sqsubseteq \sqcup X \wedge \forall y \in S : X \sqsubseteq y \Rightarrow \sqcup X \sqsubseteq y$

Аналогичным образом определяется [точная] нижняя грань

Решетки

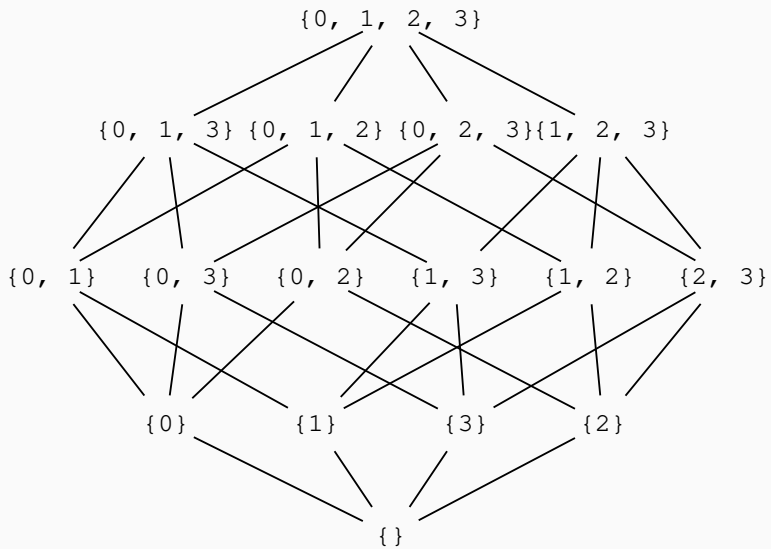
Решетка — это poset, для любых двух элементов которого есть точная верхняя и точная нижняя грани

- У решетки есть максимальный и минимальный элементы ($\top$, $\bot$)
  - Являются ли они точной верхней или нижней гранью какого-либо подмножества S?
  - Уникальны ли они?
- У решетки есть высота — длина максимально возможного пути от $\bot$ до $\top$

Интересные решетки

Булеан

- Решетка над множеством всех подмножеств S
- $\top = S$
- $\bot = \emptyset$
- $x \sqcup y = x \cup y$
- $x \sqcap y = x \cap y$



Интересные решетки

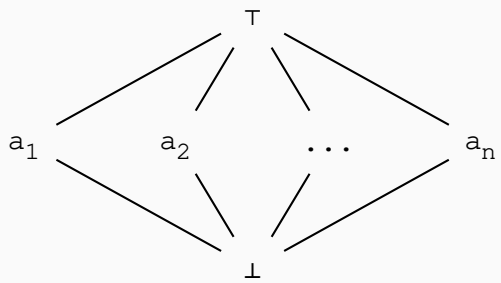
Произведение решеток

- $L_1 \times L_2 \times \ldots \times L_n = \{(x_1, x_2, \ldots, x_n) | x_i \in L_i\}$
- $(x_1, x_2, \ldots, x_n) \sqsubseteq (x'_1, x'_2, \ldots, x'_n) \Leftrightarrow \forall i : x_i \sqsubseteq x'_i$
- Как выглядит $\sqcup L_1 \times L_2 \times \ldots \times L_n$? $\sqcap L_1 \times L_2 \times \ldots \times L_n$?
- Какая высота у произведения решеток $L_1 \times L_2 \times \ldots \times L_n$?

Интересные решетки

Плоская решетка

- $flat(S) = S \cup \{\top, \bot\}$
- $\forall x, y \in S : x \sqcup y = \top \wedge x \sqcap y = \bot$



Интересные решетки

Решетка отображений

- Для множества A и решетки L решетка отображений $A \longrightarrow L$ — это решетка над функциями из элементов A в элементы L
- $A \to L = \{[a_1 \mapsto x_1, a_2 \mapsto x_2, \ldots] | A = \{a_1, a_2, \ldots\} \wedge x_1, x_2, \ldots \in L\}$
- $f \sqsubseteq g \Leftrightarrow \forall a_i \in A : f(a_i) \sqsubseteq g(a_i)$, где $f, g \in A \longrightarrow L$

А что тут с поиском точной верхней/нижней граней?

А с высотой решетки?

Зачем все это?

Анализ знака (дубль 2)

```
var a,b;
a = 42;
b = a + input;
a = a - b;
```

$$x_1 = [a \mapsto ?, b \mapsto ?]$$

$$x_2 = x_1[a \mapsto +]$$

$$x_3 = x_2[b \mapsto x_2(a) + ?]$$

$$x_4 = x_3[a \mapsto x_3(a) - x_3(b)]$$

Правила вывода (дубль 2)

Нас интересует $\sigma : [\![n]\!]$ — абстрактное состояние программы после вершины n из CFG, которое является *решеткой отображений* из переменных программы в наш абстрактный знаковый домен

$$[\![\text{var x;}]\!] = JOIN(n)[x \mapsto ?]$$
$$[\![\text{x = E;}]\!] = JOIN(n)[x \mapsto eval(JOIN(n), E)]$$
$$[\![\text{smth;}]\!] = JOIN(n)$$

$$JOIN(n) = \sqcup_{w \in pred(n)} [\![w]\!]$$

Система ограничений

- Для интересующих нас переменных $x_1, \ldots, x_n \in L$ мы получаем набор ограничений

$$
\begin{aligned}
x_1 &= f_1(x_1, x_2, \ldots, x_n) \\
x_2 &= f_2(x_1, x_2, \ldots, x_n) \\
&\ldots \\
x_n &= f_n(x_1, x_2, \ldots, x_n)
\end{aligned}
$$

Почему бы не решить его при помощи унификационного решателя?

Система ограничений

А если у меня неравенства?

$$
\begin{aligned}
x_1 &\sqsubseteq f_1(x_1, x_2, \ldots, x_n) \\
x_2 &\sqsubseteq f_2(x_1, x_2, \ldots, x_n) \\
&\ldots \\
x_n &\sqsubseteq f_n(x_1, x_2, \ldots, x_n)
\end{aligned}
$$

Можно воспользоваться тем фактом, что

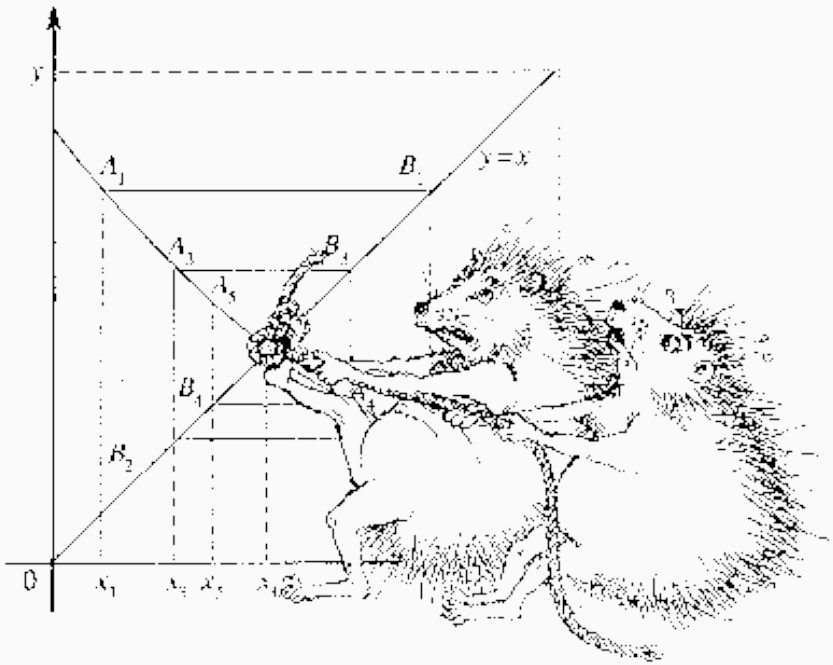
$x \sqsubseteq y \Leftrightarrow x = x \sqcap y$

Система ограничений

- Нашу систему можно упростить, если объединить f_i в одну функцию

$$f(x_1, \ldots, x_n) = (f_1(x_1, \ldots, x_n), \ldots, f_n(x_1, \ldots, x_n))$$

Надо решить $x = f(x)$



Наименьшая неподвижная точка функции

С точки зрения анализа нас интересует

наименьшее *решение* $x = f(x)$

Теорема о неподвижной точке

Для любой монотонной функции $f : L \longrightarrow L$, определенной на решетке L конечной длины, существует наименьшая неподвижная точка вида

$$fix(f) = \sqcup_{i \geqslant 0} f^{\,i}(\bot)$$

Монотонные функции

- Функция $f : L \longrightarrow L$ является монотонной на решетке L, если $\forall x, y \in L : x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y)$
- Функция нескольких аргументов является монотонной, если она монотонна по каждому аргументу
- Композиция сохраняет монотонность
    - Можно попробовать доказать этот факт на досуге

Монотонность анализа знака

$$[\![\text{var x;}]\!] = JOIN(n)[x \mapsto ?]$$
$$[\![\text{x = E;}]\!] = JOIN(n)[x \mapsto eval(JOIN(n), E)]$$
$$[\![\text{smth;}]\!] = JOIN(n)$$

$$JOIN(n) = \sqcup_{w \in pred(n)} [\![w]\!]$$

- Монотонность $\sqcup$?
- Монотонность $\sigma[a \mapsto v]$?
- Монотонность *eval*? Абстрактных операторов?

Монотонность абстрактных операторов

- Медленно и неспешно проверить вручную
- Перебрать все варианты для наших таблиц

$$\forall x, y, x' \in L : x \sqsubseteq x' \Rightarrow x \mathbin{\overline{op}} y \sqsubseteq x' \mathbin{\overline{op}} y$$
$$\forall x, y, y' \in L : y \sqsubseteq y' \Rightarrow x \mathbin{\overline{op}} y \sqsubseteq x \mathbin{\overline{op}} y'$$

Монотонный фреймворк

- CFG для анализируемой программы в виде вершин $Nodes = \{v_1, \ldots, v_n\}$
- Решетка конечной длины L, представляющая интересующий нас абстрактный домен
- Набор переменных $[\![v_i]\!] \in L$ для каждой вершины CFG
- Ограничения для различных видов узлов CFG
  - Определяют значение $[\![v_i]\!]$ через другие переменные
  - Часто учитывают структуру CFG
  - Должны быть монотонными

Монотонный фреймворк

Как его применять на практике?

- Извлечь все ограничения из CFG
- Решить их, используя алгоритм поиска неподвижной точки
  - Работает над решеткой L^n
  - Потому что неподвижная точка ищется для объединенной функции

$$f(x_1, \ldots, x_n) = (f_1(x_1, \ldots, x_n), \ldots, f_n(x_1, \ldots, x_n))$$

Алгоритмы поиска неподвижной точки

Базовый алгоритм

function NaiveFixPointAlgorithm(f)
  $x \leftarrow (\bot, \bot, \ldots, \bot)$
  repeat
    $x \leftarrow f(x)$
  until $x = f(x)$
  return x
end function

- Корректен по построению
- Никак не использует структуру программы или решеток

Алгоритм round-robin

- Одна из проблем базового алгоритма — медленное распространение обновленных значений

Алгоритм round-robin

- Одна из проблем базового алгоритма — медленное распространение обновленных значений

function RoundRobinFixPointAlgorithm(f)
  $(x_1, x_2, \ldots, x_n) \leftarrow (\bot, \bot, \ldots, \bot)$
  repeat
    for $i \leftarrow 1 \ldots n$ **do**
      $x_i = f_i(x_1, x_2, \ldots, x_n)$
    end for
  until $(x_1, x_2, \ldots, x_n) = f(x_1, x_2, \ldots, x_n)$
  return x
end function

Корректен ли предложенный алгоритм?

Алгоритм хаотических итераций

- Если подумать, можно понять, что порядок не особо и важен

function ChaoticFixPointAlgorithm(f)
  $(x_1, x_2, \ldots, x_n) \leftarrow (\bot, \bot, \ldots, \bot)$
  repeat
    $i \leftarrow$ *RandomOf(1 ...n)*
    $x_i = f_i(x_1, x_2, \ldots, x_n)$
  until $(x_1, x_2, \ldots, x_n) = f(x_1, x_2, \ldots, x_n)$
  return x
end function

Структурный алгоритм

- Если подумать еще чуть-чуть, можно вспомнить, что наши уравнения строятся по CFG программы
- Зависимости между уравнениями часто соответствуют зависимостям по CFG
- Этим фактом можно воспользоваться!

$$deps : \text{Nodes} \longrightarrow 2^{\text{Nodes}}$$

function SimpleWorkListFixPointAlgorithm(f)

$(x_1, x_2, \ldots, x_n) \leftarrow (\bot, \bot, \ldots, \bot)$

$WL \leftarrow \{v_1, v_2, \ldots, v_n\}$

while $WL \neq \emptyset$ do

$\{v_i\} \cup WL \leftarrow WL$

$y = f_i(x_1, x_2, \ldots, x_n)$

if $x_i \neq y$ then

$x_i \leftarrow y$

$WL \leftarrow WL \cup deps(v_i)$

end if

end while

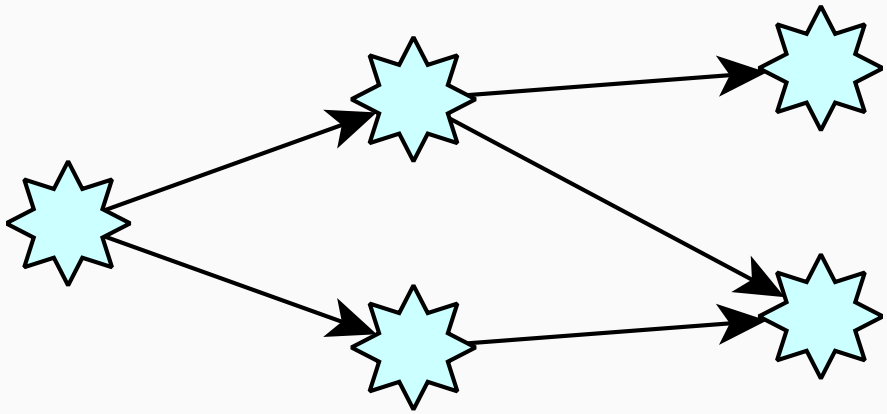
return x

end function

Алгоритмы поиска неподвижной точки

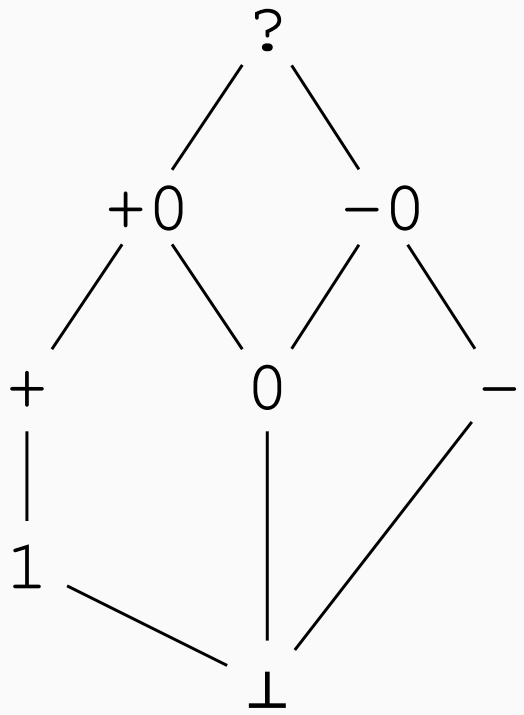
Можно еще лучше!

- Узлы CFG могут иметь приоритет на обработку
- Можно рассмотреть граф зависимостей и найти в нем сильно связанные компоненты
    - Зависимости в полученном графе можно решать снизу вверх



Монотонный фреймворк

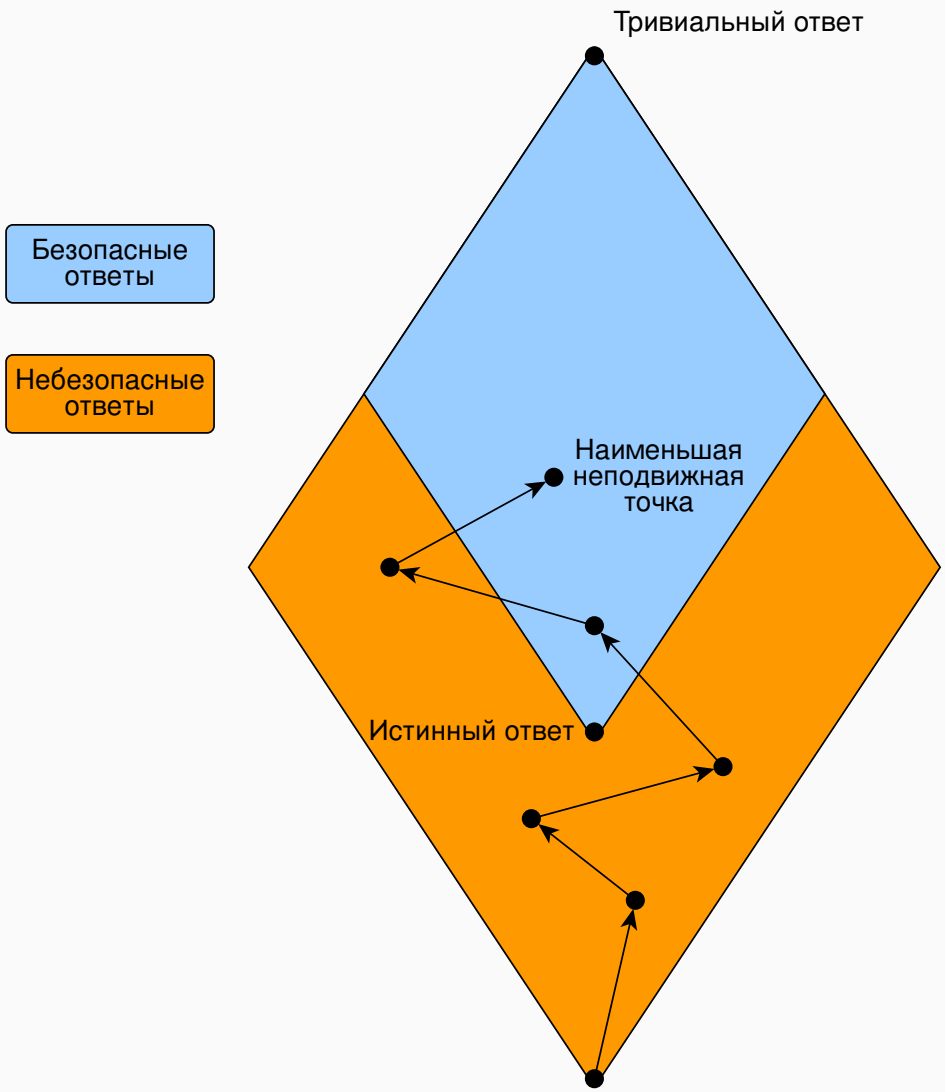
Почему это работает?



- $x \sqsubseteq y$ значит

  «x как минимум такой же точный, как y»
- Наша функция f пытается улучшить результат
- Монотонность f значит, что мы никогда не ухудшаем результат для более точных входных данных

Монотонный фреймворк



Must vs May

Что такое безопасный ответ?

- Для анализа знака — ответ, которые точно включает в себя истинный ответ
  - *Пере-аппроксимация* aka *may-анализ*
  - Идем по решетке снизу вверх, от частных к общим результатам

Must vs May

А может быть, все было наоборот?

- Если нам интересна *недо-аппроксимация* aka *must-анализ*?
  - Например, для анализа инициализации
  - Идем по решетке сверху вниз, от общих к частным результатам

На практике для must-анализа просто рассматривают решетку с отношением $\sqsupseteq$

На подумать

- Вопросы, раскиданные по лекции
- Можно ли выразить анализ типов с предыдущей лекции как анализ над решетками?
    - Если да, то как выглядит наша решетка?
    - Если нет, то почему?
- Можно ли выразить анализ над решетками как анализ типов?
    - Короткий ответ: да, можно (subtyping)
    - Длинный ответ: подумайте, все ли выразимо в таком случае

На поделать

- Допишите метод `transfer`
  в трейте `IntraprocSignAnalysisFunctions`[2]
- Реализуйте класс `PowersetLattice`
  в файле `GenericLattices`[3]

  Все реализации — очень и очень простые

[2] src/tip/analysis/SignAnalysis.scala

[3] src/tip/lattices/GenericLattices.scala

В следующей серии

Как вся эта теоретическая ерунда работает на практике?

akhin@kspt.icc.spbstu.ru

belyaev@kspt.icc.spbstu.ru