Дискретная математика

Абстрактная интерпретация

Константин Чухарев

Осень 2026

Абстрактная интерпретация

> *Все модели неверны, но некоторые полезны.*
>
> ***— Джордж Бокс***

Почему тестов недостаточно

Программа, делящая на введённое число, прошла тесты на входах $x = 5$, $x = -2$, $x = 7$. Все три запуска успешны — и всё равно ничего не доказано. На четвёртом входе, $x = 0$, программа упадёт с ошибкой деления на ноль. Тестирование показывает наличие ошибок, но никогда не доказывает их отсутствия: ошибка может прятаться на любом следующем входе.

Перебрать все входы невозможно. Вход — 64-битное целое: возможных значений 2^{64}, больше, чем секунд в возрасте Вселенной. Для программы, читающей список чисел, входов бесконечно много.

> Доказывать свойство нужно для всех запусков сразу, не выполняя программу ни на одном из них.

Аппроксимация вместо точного ответа

Точный автоматический ответ недостижим: свойства произвольной программы неразрешимы. Остаётся приближённый ответ — и его оказывается достаточно. Точный ответ «да» или «нет» гарантировать нельзя. Достаточно разрешить третий ответ, «не знаю», и задача перестаёт быть безнадёжной.

Простейший корректный анализ отвечает «не знаю» на любой вопрос: он не ошибается, но и не сообщает ничего полезного. Вся дальнейшая работа — ужимать этот «не знаю» до точного ответа там, где это возможно.

> Аппарат для приближённого ответа уже знаком: знаковая решётка, супремум и неподвижная точка. Вместо точных значений анализ работает с их грубыми обобщениями — *абстракциями*. Цена — неточность: анализ может сообщить о возможной ошибке там, где на деле всё в порядке. Корректность при этом сохраняется: ни одна реальная ошибка не пропадает.

Знаки вместо чисел

Идея проста: заменить точные значения их грубыми обобщениями. Чтобы доказать, что знаменатель не равен нулю, достаточно знать, что x не бывает нулём. Вместо числа возьмём его *знак*: положительное, отрицательное или ноль. Добавим два особых значения — «неизвестно» и «недостижимо». Таких обобщений конечное число, и программа «выполняется» на них за конечное число шагов.

Пример

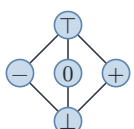
Для доказательства отсутствия деления на ноль в программе `y := 10 / x` достаточно знать знак x. Домен знаков различает случаи: при $x \neq 0$ деление на ноль невозможно. При $x = 0$ — возможно.

Домен знаков

Определение 1 (Домен знаков)

Домен знаков состоит из пяти элементов:

- $\top$ — «знак неизвестен» (значение может быть любым),
- $+$ — «значение положительно»,
- 0 — «значение равно нулю»,
- $-$ — «значение отрицательно»,
- $\bot$ — «значение невозможно» (переменная здесь не определена).



Элементы упорядочены по точности: $\bot$ — самое точное (значения нет), $\top$ — самое грубое (значение любое). Нагляднее всего порядок виден на решётке: $\bot \preceq \{-, 0, +\} \preceq \top$.

Информационный порядок

Определение 2 (Информационный порядок)

Говорим, что $a \preceq b$ («a не грубее b»), если любой конкретный смысл a содержится в конкретном смысле b. Например, $+ \preceq \top$: «положительное» точнее, чем «любое». И $\bot \preceq +$: «недостижимо» точнее, чем «положительное».

Почему порядок «наоборот»

Абстракция и конкретизация

Каждое абстрактное значение описывает множество конкретных значений. Значение $+$ описывает все положительные числа, $\top$ — все целые, 0 — только ноль.

Определение 3 (Конкретизация)

Пусть $\mathcal{C}$ — множество конкретных состояний программы, а D — абстрактный домен. **Конкретизация** $\gamma : D \to \mathcal{P}(\mathcal{C})$ отображает абстрактное значение в множество конкретных состояний, которые оно представляет.

Определение 4 (Абстракция)

Абстракция $\alpha : \mathcal{P}(\mathcal{C}) \to D$ отображает множество конкретных состояний в наиболее точное абстрактное значение, его покрывающее:

$$\alpha(X) = \sqcap \{d \in D \mid X \subseteq \gamma(d)\}.$$

Для домена знаков $\alpha(\{5, 7\}) = +$. Конкретизация: $\gamma(+) = \{x \in \mathbb{Z} \mid x > 0\}$.

Связка Галуа

Связка Галуа

Абстрактный домен

Определение 5 (Абстрактный домен)

Абстрактный домен — решётка $(D, \preceq, \bot, \top, \sqcup, \sqcap)$, где:

- $\preceq$ — информационный порядок («не грубее»),
- $\bot$ — наименьший элемент («недостижимо»),
- $\top$ — наибольший элемент (все значения),
- $\sqcup$ — объединение: $a \sqcup b$ описывает всё, что описывают a и b,
- $\sqcap$ — пересечение: $a \sqcap b$ — наиболее точное значение, совместимое с обоими.

Объединение соответствует слиянию веток программы, пересечение — усилению требования.

Абстрактные состояния

Для программы с n переменными абстрактное состояние — вектор из n абстрактных значений. Например, $(-, +, 0, ...)$. Порядок поэлементный: $a \preceq b \Leftrightarrow \forall i. a_i \preceq b_i$. Конечный текст программы описывается конечным множеством абстрактных состояний.

Переносящие функции

Конкретная семантика оператора превращает множество состояний в множество состояний. Например, оператор $x := x + 1$ переводит множество значений в сдвинутое множество. То есть $X + 1 = \{v + 1 \mid v \in X\}$.

Определение 6 (Переносящая функция)

Переносящая функция — отображение $[\![s]\!]^\sharp : D \to D$, которое вычисляет абстрактное состояние после оператора s по абстрактному состоянию до оператора. Интуиция: «запустить оператор на абстрактных значениях».

Пример: присваивание в домене знаков

Пример (Присваивание в домене знаков)

Пусть $x := x + 1$. Абстрактный результат:

- $+ \to +$ — положительное плюс единица остаётся положительным;
- $- \to \top$ — отрицательное плюс единица может стать нулём;
- $0 \to +$;
- $\top \to \top$;
- $\bot \to \bot$.

Правило $- \to \top$ — цена абстракции: знак не различает -1 и -5. Он не видит, что $-1 + 1 = 0$.

Корректность переноса

Переносящая функция обязана быть *корректной*: она не имеет права терять ни одно реальное поведение.

Определение 7 (Корректность переноса)

Переносящая функция $[\![s]\!]^\sharp$ *корректна*, если для любого абстрактного состояния d

$$[\![s]\!](\gamma(d)) \subseteq \gamma\big([\![s]\!]^\sharp(d)\big),$$

где $[\![s]\!]$ — конкретная семантика оператора s на множествах состояний. Перенос добавляет лишь невозможные поведения и не теряет настоящие.

Определение 8 (Монотонность)

Перенос $[\![s]\!]^\sharp$ *монотонен*, если из $d_1 \preceq d_2$ следует $[\![s]\!]^\sharp(d_1) \preceq [\![s]\!]^\sharp(d_2)$. Без монотонности итерации по циклу могут не сходиться.

Сложение в домене знаков

Пример (Сложение в домене знаков)

Операция $z := x + y$ требует абстрактной таблицы сложения знаков. Правила:

- $+ \boxplus + = +$;
- $- \boxplus - = -$;
- $x \boxplus 0 = x$ для любого знака x;
- $+ \boxplus - = \top$ — сумма знаков может быть любой;
- $\top \boxplus x = \top$ при $x \neq \bot$;
- $\bot \boxplus x = \bot$.

Правило $+ \boxplus - = \top$ — источник неточности: знак не хранит величин. Он не видит, что $7 + (-4) = 3 > 0$. Корректность сохраняется: абстрактный результат $\top$ покрывает реальный.

Интервальный домен

Программа суммирует элементы массива:

```
s := 0
for i in 0..n do
  s := s + a[i]
```

Индекс i не должен выйти за границы массива. Чтобы доказать это, достаточно знать, что i лежит в отрезке $[0, n-1]$.

Определение 9 (Интервальный домен)

Абстрактное значение — интервал $[l, h]$ либо особый элемент $\bot$. Здесь $l, h \in \mathbb{Z} \cup \{-\infty, +\infty\}$. Конкретизация — все целые числа из интервала $[l, h]$. $\top = [-\infty, +\infty]$. Порядок по включению: $[a, b] \preceq [c, d] \Leftrightarrow [a, b] \subseteq [c, d]$. Объединение — выпуклая оболочка: $[a, b] \sqcup [c, d] = [\min(a, c), \max(b, d)]$. Пересечение — наиболее точный интервал: $[\max(a, c), \min(b, d)]$ (пусто, если нижняя граница больше верхней).

Интервалы и связи между переменными

Сложение интервалов поэлементное: $[a, b] + [c, d] = [a + c, b + d]$.

Интервальный домен теряет связи между переменными. Если в каждой точке программы $y = x - 1$, точное множество пар лежит на прямой. Интервальный анализ видит лишь прямоугольник $x \in [2, 5], y \in [1, 4]$ и не замечает, что y определён значением x. Связи между переменными возвращают более богатые домены — октагоны и полиэдры.

Пример (Выход за границы массива)

Пусть в программе $i \in [0, 9]$, а массив имеет 10 элементов. Перенос обращения к элементу $A[i]$ проверяет, что индекс в границах. Если интервал i выходит за границы — например, $[0, 10]$ — анализатор сообщает о возможном выходе за границы. Сообщение может оказаться ложным, а реальный выход за границы он не пропустит.

Домен констант

Определение 10 (Домен констант)

Абстрактное значение — либо конкретное целое c, либо особое значение $\top$ («не константа»), либо $\bot$ («недостижимо»). Порядок: $\bot \preceq c \preceq \top$ для всех целых c.

Домен констант доказывает равенства. После присваиваний $x := 3, y := x + 4$ выполняется равенство

$$y = 7.$$

Компиляторы используют его для *распространения констант*: подставить вычисленное значение и упростить выражение.

Пример: распространение констант

Пример (Распространение констант)

Программа:

```
x := 3
y := x + 4
z := y * 2
```

Анализ констант: $x = 3, y = 7, z = 14$. Компилятор заменит $z := y \cdot 2 \to z := 14$.

Перенос условий

Определение 11 (Перенос условий)

Условие переносится как *фильтр*: абстрактное состояние пересекается с множеством, заданным условием. Например, для условия `i < n` интервал i в ветке «да» становится $[l, h] \to [l, \min(h, n-1)]$ (при известной константе n). Если $l > n - 1$ — ветка «да» недостижима. Ветка «нет» получает пересечение с дополнением: $[l, h] \sqcap [n, +\infty]$. Если n неизвестна (её интервал — $\top$), фильтр ничего не отсекает.

Неподвижная точка цикла

Простая последовательность операторов переносится естественно: идём по программе, применяя переносящие функции. С условными операторами объединяем ветки через $\sqcup$. Сложность — циклы: тело цикла может выполняться сколько угодно раз.

Пусть E — состояние в голове цикла при входе. Определим $F(d) = E \sqcup [\![B]\!]^\sharp(d)$: объединение состояния на входе и результата одного прохода тела из d. Состояние после любого числа проходов — наименьшая неподвижная точка F:

$$\mathrm{lfp}(F) = \sqcup_{k \geq 0} F^k(\bot).$$

Смысл: последовательность $F^k(\bot)$ монотонно накапливает состояния, достижимые в голове цикла. Объединив все $F^k(\bot)$, получаем состояние после любого числа проходов.

Итерация Клини

Теорема 1 (Итерация Клини)

Если F монотонна и домен конечен, последовательность итераций $d_0 = \bot, d_{k+1} = F(d_k)$ достигает $\mathrm{lfp}(F)$ за конечное число шагов.

Доказательство (Доказательство)

Монотонность даёт $d_0 \preceq d_1 \preceq d_2 \preceq \ldots$: последовательность не убывает. В конечной решётке любая неубывающая последовательность стабилизируется. С какого-то момента $d_{k+1} = d_k$. Тогда $F(d_k) = d_k$ — неподвижная точка. Она наименьшая: любая неподвижная точка содержит $\bot = d_0$. Значит, она содержит и все последующие d_k.

Расходящаяся итерация

Домен знаков конечен — циклы в нём анализируются перебором. Интервальный домен бесконечен: интервалы вида $[0, n]$ с ростом n никогда не стабилизируются. Именно здесь наивная итерация ломается.

Пример (Расходящаяся итерация)

Рассмотрим цикл

```
i := 0
while i < n do
  i := i + 1
```

Интервальный анализ состояния i в голове цикла даёт последовательность $[0, 0], [0, 1], [0, 2], [0, 3], \ldots$. Каждая итерация расширяет интервал на единицу. Процесс никогда не остановится: наивной итерацией неподвижную точку не получить.

Расходящаяся итерация [2]

Условие `i < n` здесь не помогает: значение n неизвестно (его интервал — $\top$), поэтому фильтр условия ничего не отсекает.

Widening

Определение 12 (Оператор расширения)

Бинарная операция $\nabla : D \times D \to D$, для которой:

- $a \sqcup b \preceq a \nabla b$ (результат не менее груб, чем объединение),
- для любой возрастающей последовательности $d_0 \preceq d_1 \preceq \ldots$ последовательность $w_0 = d_0, w_{k+1} = w_k \nabla d_{k+1}$ стабилизируется за конечное число шагов.

Словами: оператор ∇ «ускоряет» рост, сбрасывая нестабильные компоненты в $\top$.

Widening для интервалов

Пример (Widening для интервалов)

Классический widening:

$$[l_1, h_1] \nabla [l_2, h_2] = [l, h],$$

$$l = \begin{cases} -\infty \text{ если } l_2 < l_1 \\ l_1 \text{ иначе} \end{cases},$$

$$h = \begin{cases} +\infty \text{ если } h_2 > h_1 \\ h_1 \text{ иначе} \end{cases}.$$

В примере со счётчиком $[0, 0] \nabla [0, 1] = [0, +\infty]$: верхняя граница выросла — сбрасываем её в бесконечность. Следующая итерация: $[0, +\infty] \nabla [0, +\infty] = [0, +\infty]$ — стабилизация. Неподвижная точка достигнута за два шага.

Narrowing

Определение 13 (Оператор сужения)

Бинарная операция $\triangle : D \times D \to D$. Результат не грубее первого операнда: $a \triangle b \preceq a$. Повторяя сужение из уже достигнутой неподвижной точки, получаем более точную неподвижную точку.

Пример (Narrowing в примере со счётчиком)

Сужение идёт по условию цикла. При известной константе n фильтр `i < n` уточняет интервал $[0, +\infty] \to [0, n-1]$. Верхняя граница восстановлена из условия.

Схема «widening для сходимости, narrowing для точности» — стандартный рецепт анализа циклов в интервальном домене.

Цена аппроксимации

Абстрактный анализ корректен, но не точен. Корректность здесь означает: анализатор не пропускает реальные ошибки. Неточность означает: он может сообщить об ошибке, которой на деле нет.

Определение 14 (Ложное срабатывание)

Если анализатор сообщает о возможной ошибке в точке, где ошибки не бывает, — это *ложное срабатывание* (false positive). Корректный анализатор допускает ложные срабатывания, но не допускает пропусков.

Абстракция рисует вокруг исполнений оболочку. Если оболочка не задевает «запретную» область, ошибка доказанно невозможна. Если задевает — анализатор не различает реальную ошибку и ложную тревогу. Замалчивание в неоднозначном случае было бы пропущенной реальной ошибкой.

Пример: точность и домен

Пример (Точность и домен)

Рассмотрим программу $x := 6, y := -7, z := x + y$. Домен знаков даёт для неё $z = \top$: знак суммы не вычисляется. В действительности $z = -1$ — определённое число. Домен констант посчитал бы $z = -1$ точно. Домен интервалов — $[-1, -1]$. Выбор домена — компромисс между точностью, стоимостью и свойством, которое нужно доказать.

Применения

Абстрактная интерпретация — основа индустрии статического анализа.

- Оптимизирующие компиляторы (GCC, LLVM) внутри — анализаторы потоков данных: распространение констант, анализ диапазонов, удаление мёртвого кода.
- Astrée доказала отсутствие ошибок времени выполнения в полётном софте Airbus A340 и A380.
- Polyspace (MathWorks) верифицирует встраиваемый софт.
- Frama-C — открытая платформа анализа кода на C.

> Абстрактная интерпретация вычисляет инварианты циклов автоматически. Человеку их придумывать не нужно.

Символьное исполнение

Символьное исполнение — противоположный полюс: оно перебирает пути точно, ценой стоимости. На практике подходы работают вместе: анализ находит подозрительные места, символьное исполнение подтверждает их по конкретным путям.

Что мы поняли?

- Тестирование не доказывает отсутствие ошибок: свойство нужно доказывать для всех запусков сразу.
- Точный анализ программ неразрешим — работает корректная аппроксимация с третьим ответом «не знаю».
- Абстрактный домен — решётка с информационным порядком; связь с конкретным задают α и γ.
- Переносящие функции переносят операторы в абстрактный мир и обязаны быть корректными и монотонными.
- Знаки, интервалы и константы — домены для разных свойств.
- Неподвижная точка переносящей функции цикла — результат анализа; widening и narrowing обеспечивают сходимость.
- Ложные срабатывания неизбежны, пропуски ошибок — нет.

Вопросы для самопроверки

- Почему тестирование не может доказать отсутствие ошибок?
- Что такое абстрактный домен, и зачем нужны $\bot$ и $\top$?
- Как связаны абстракция α и конкретизация γ?
- Почему $+ \boxplus - = \top$ в домене знаков?
- Чем интервальный домен отличается от знакового?
- Почему наивная итерация для интервалов расходится, и как widening это исправляет?
- Почему ложные срабатывания неизбежны, а пропуски ошибок — нет?