Анализ кода и информационная безопасность

Лекция 05

МГУ / ВМК / СП

0501

АБСТРАКТНАЯ ИНТЕРПРЕТАЦИЯ ТЕОРИЯ

Абстрактная интерпретация

- Абстрактная интерпретация – теория корректной аппроксимации семантики программы.
- Задача абстрактной интерпретации – построить корректные абстракции для анализа программы.
- **Важно**: одни и те же абстракции можно применять при проведении различных анализов.

Модельный язык

- В качестве модельного языка рассматривается подмножество языка C:
  - **Нет указателей, все массивы глобальные заданной размерности**
  - **Все переменные имеют либо тип int, либо int [N], где N - константа**
  - **Программа состоит из одной функции**
  - **Все локальные переменные объявлены в начале функции и инициализированы нулём.**

Конкретные состояния

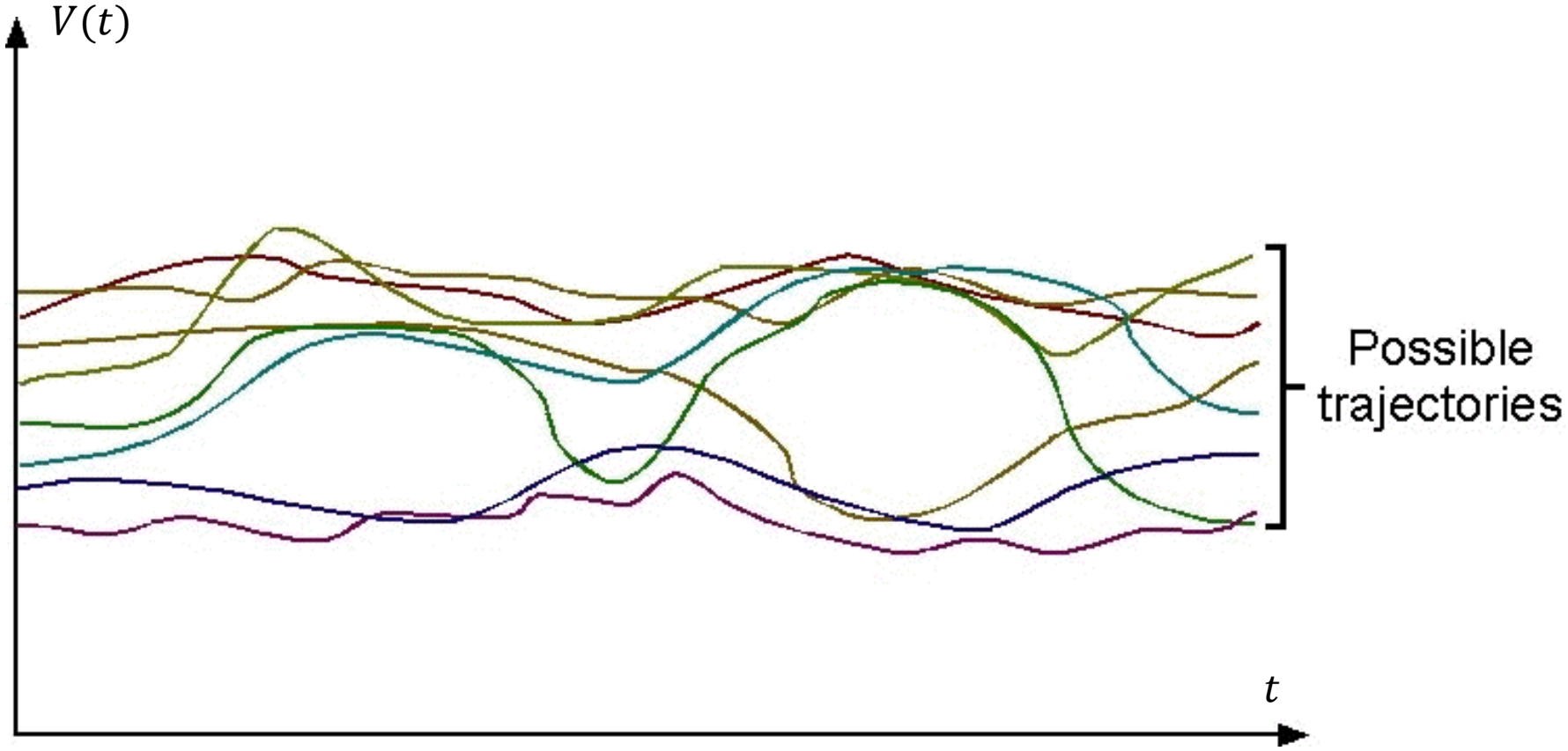
```
1) int gcd(int a, int b) {   // a = 45, b = 10, x = 0
2)   int x;                 // a = 45, b = 10, x = 0
3)   while (b != 0) {        // a = 45, b = 10, x = 0
4)     x = a % b;           // a = 45, b = 10, x = 5
5)     a = b;               // a = 10, b = 10, x = 5
6)     b = x;               // a = 10, b = 5, x = 5
7)   }
8)   return a;
9) }
```

Конкретное состояние программы – значения её переменных.

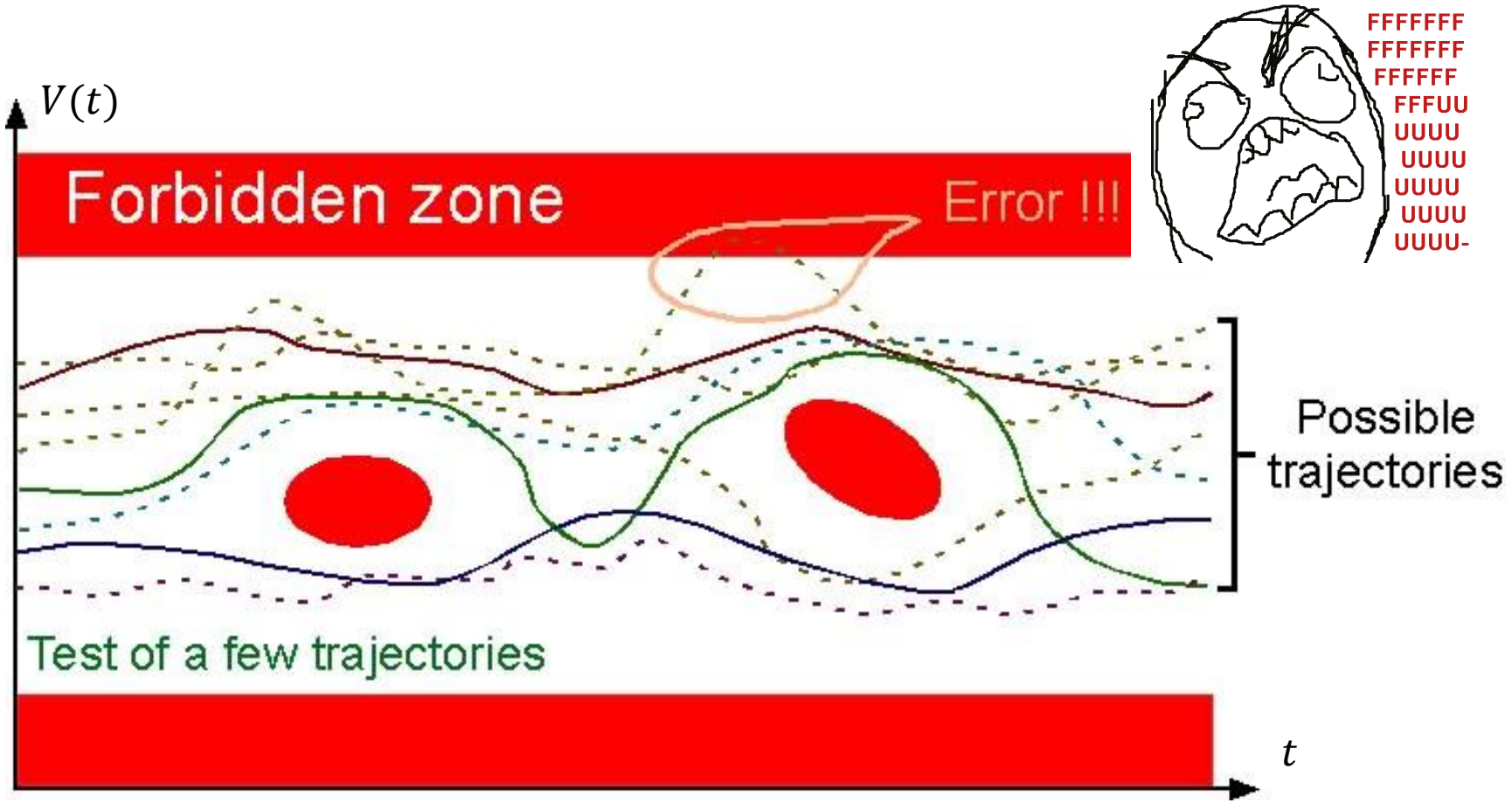
P. Cousot and R. Cousot

Astree: верификация программ основанная на абстрактной интерпретакции.

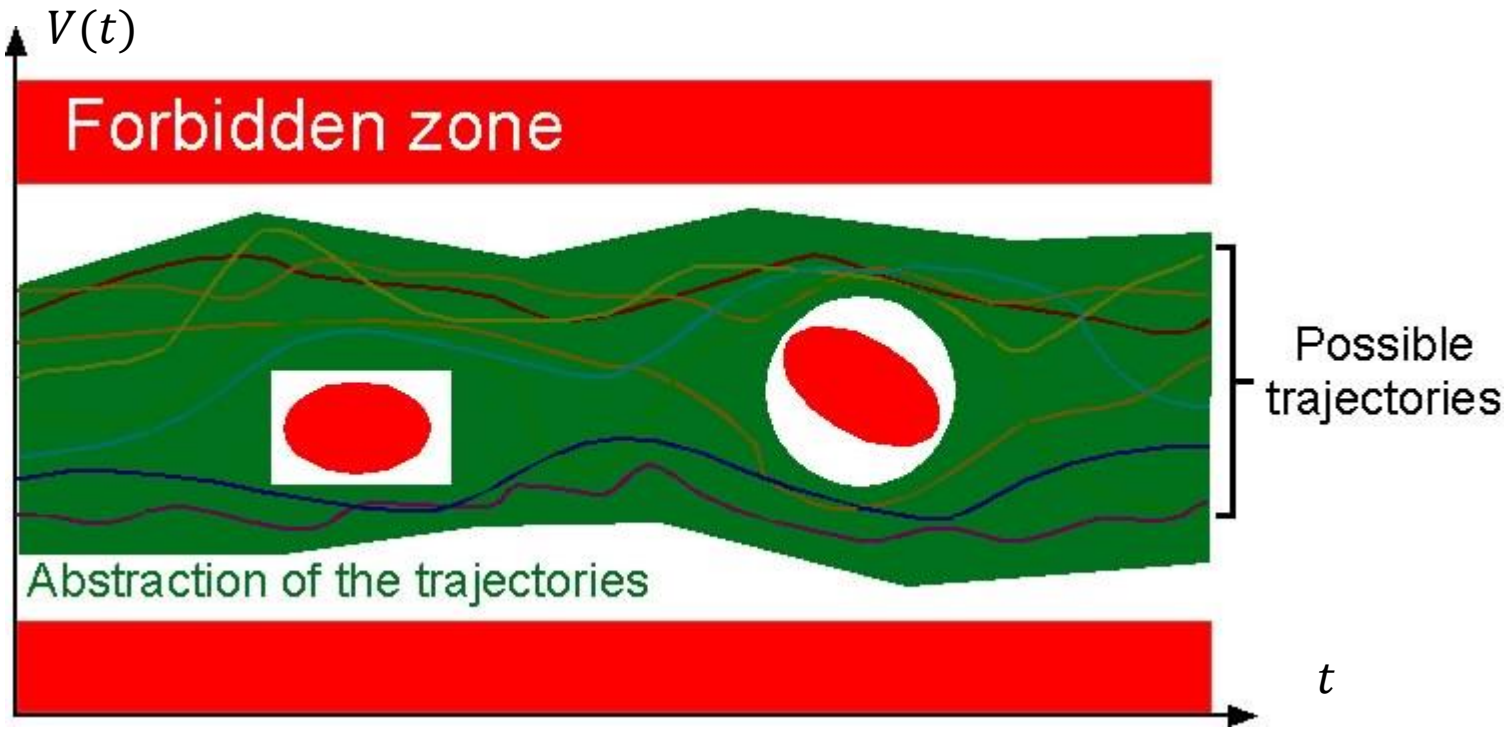
Конкретные выполнения



Тестирование



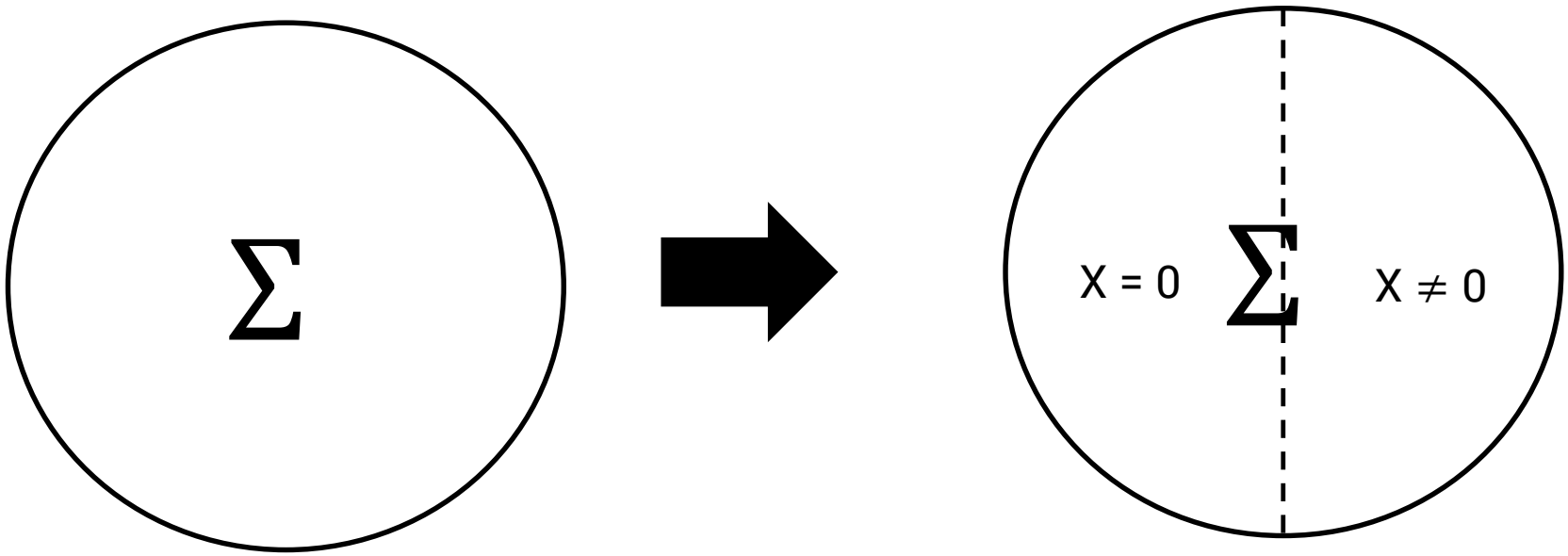
Абстрактная интерпретация



≠

Абстрактные состояния

Σ — множество всех конкретных состояний
x = 0, x ≠ 0 — абстрактные состояния



Абстрактные состояния

- Абстрактное состояние — подмножество Σ
- Примеры:
  - $x = 0$, *множество конкретных состояний, удовлетворяющие уравнению* $x = 0$.
  - $x \in [10, 20], y \in [100, 200]$
  - $x = 0 \vee y = 3 \wedge x > y$
- L – множество абстрактных состояний, $L \subseteq 2^{\Sigma}$

Решетки с верхним и нижним элементами

Множество L с двумя бинарными операциями ∧ (сбор) и ∨ (объединение) называется решёткой, если верно, что:

1. $a \vee a = a, \quad a \wedge a = a$ (идемпотентность)
2. $a \vee b = b \vee a, \quad a \wedge b = b \wedge a$ (коммутативность)
3. $a \vee (a \wedge b) = a \wedge (a \vee b)$ (поглощение)
4. $(a \vee b) \vee c = a \vee (b \vee c)$,
   $(a \wedge b) \wedge c = a \wedge (b \wedge c)$ (ассоциативность)

Запись $a \leq b$ означает $a \vee b = b$ $(\text{или } a \wedge b = a)$

Дополнительно потребуем существования элементов ⊤, ⊥, таких что: $\bot \leq a \leq \top$

Решетка 2^{Σ}:

- 2^{Σ} является решеткой:
  - В качестве операции сбора $(\wedge)$ – используем пересечение множеств.
  - В качестве операции объединения $(\vee)$ – слияние множеств.
  - Порядок в 2^{Σ} определяется по включению.
  - Нижним элементом является пустое множество
  - Верхним элементом является множество Σ

Требования к множество абстрактных состояний

- Множество абстрактных состояний должно являться решеткой с верхним и нижним элементом.

- **Важно:** операции сбора и объединения могут не совпадать с пересечением и объединением соответствующих множеств конкретных состояний.

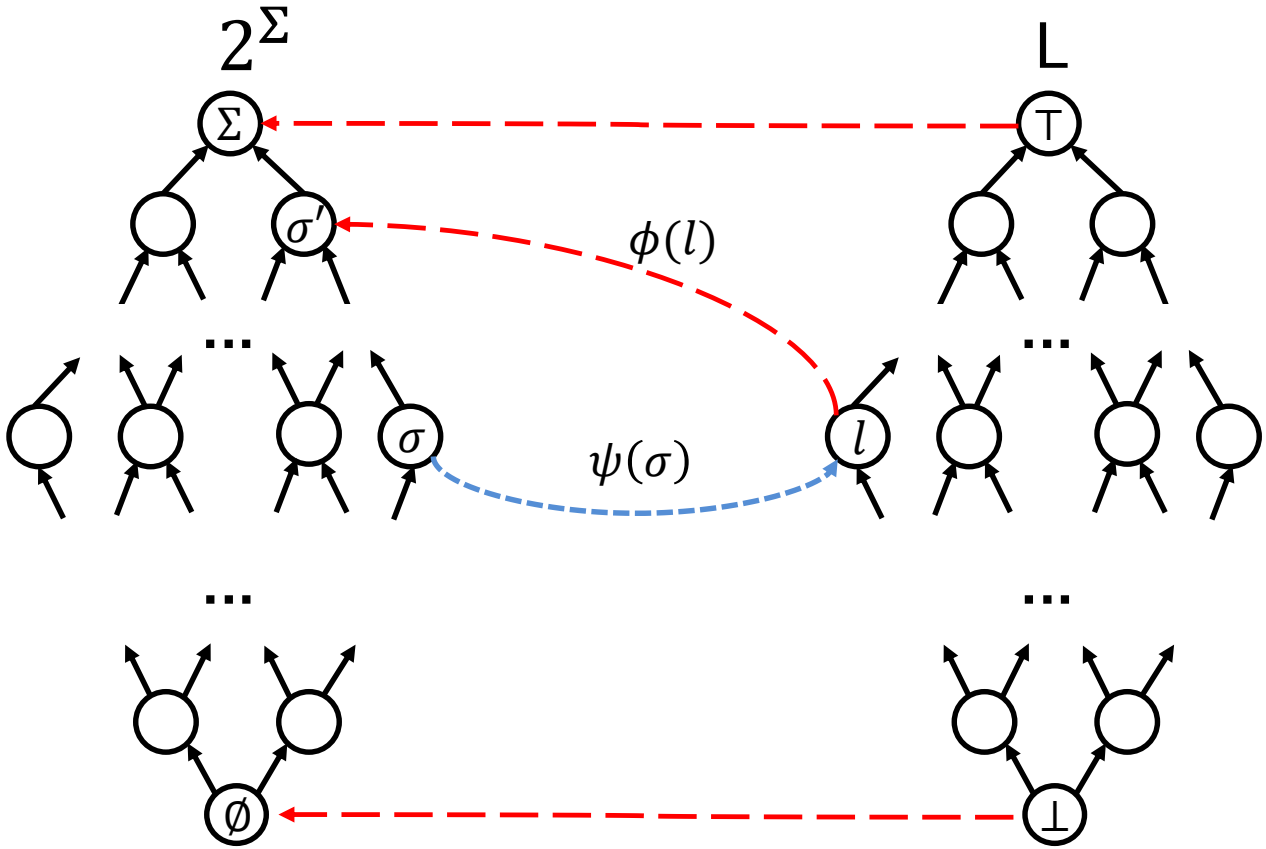
- Задано соответствие между L и 2^{Σ}, а именно:

  конкретизация: $\phi(l) = \sigma, \text{где } l \in L, \sigma \in 2^{\Sigma}$

  абстракция: $\psi(\sigma) = l, \text{где } l \in L, \sigma \in 2^{\Sigma}$
- $\phi(l)$ задаётся по построению
- $\psi(\sigma)$ определяется как самое точное l, такое что:

  $$\sigma \in \phi(l)$$
- $\phi(l)$ и $\psi(\sigma)$ задают соответствие Галуа: $\sigma \leq \phi\big(\psi(\sigma)\big)$

Соотношение между L и 2^{Σ}



Передаточные функции

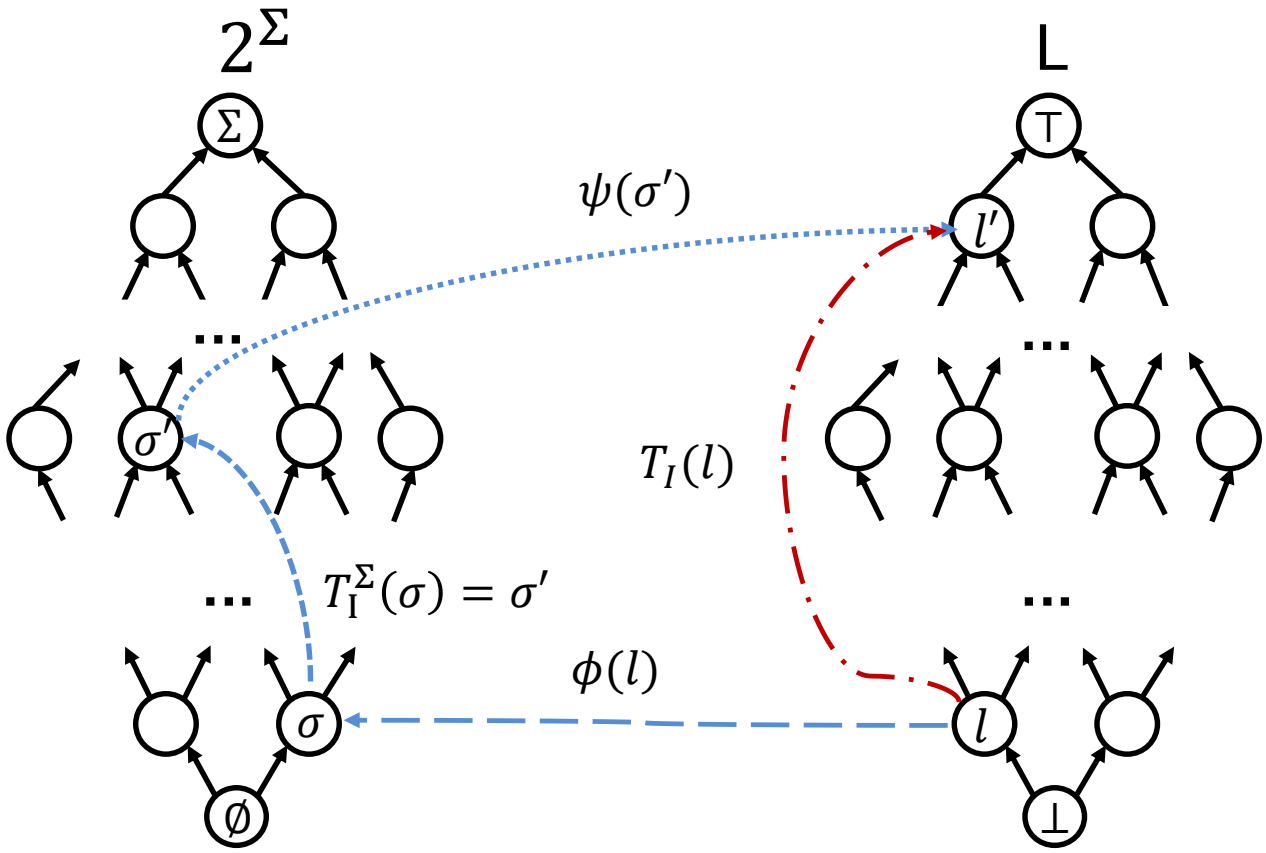
- Передаточной функцией для инструкции I будем называть отображение $\mathrm{T_I} : L \to L$, задающее семантику инструкции I для абстрактных состояний.

- Построим точную передаточную функцию при помощи функций соответствия.

- Пусть $T_I^{\Sigma} : 2^{\Sigma} \to 2^{\Sigma}$, передаточная функция заданная для 2^{Σ} в соответствии с семантикой I, тогда:

$$T_I(l) = \psi(\mathrm{T}_I^{\Sigma}(\phi(l)))$$

Передаточные функции



Свойства построенной передаточной функции

- Данная передаточная функция является самой точной для данного множества абстрактных состояний.

- Используемые на практике передаточные функции могут быть менее точными, чем T_I.

- Пусть T_I' произвольная функция из $L \to L$, тогда для того, чтобы её можно было бы использовать в качестве передаточной функции, необходимо чтобы:

$$T_I(l) \subseteq T_I'(l)$$

Использование менее точных передаточных функций называется **widening**. Оно применяется для ускорения сходимости анализа потока данных.

Применение абстрактной интерпретации

- Вычисление необходимых условий для каждой точки программы
  - Поиск недостижимого кода
  - Поиск константных выражений

- Символьное выполнение программы
  - Поиск переполнения буфера
  - Поиск разыменования нулевого указателя

Вычисление необходимых условий

- Заметим, что 2^{Σ} и T_I^{Σ} могут быть использованы при анализе потока данных

- Тогда MOP-решение для заданного базового блока B даст набор всех возможных конкретных состояний, с которыми программа может оказаться в данном базовом блоке. Обозначим его P_B, $P_B \in 2^{\Sigma}$.

- Тогда необходимым условием для B будем называть такое абстрактное состояние l, что $P_B \subseteq \phi(l)$.

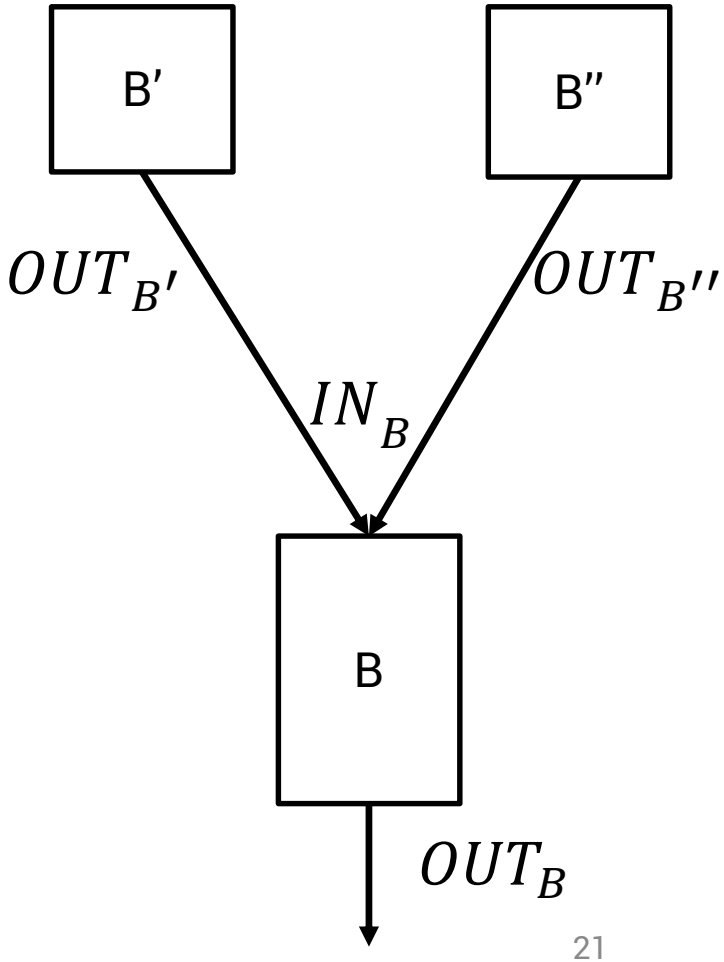
Уравнения для нахождения необходимого условия

Система уравнений:

Для всех B: $\begin{cases} IN_B = \bigwedge_{\{B',B\}} OUT_{B'} \\ OUT_B = T_B(IN_B) \end{cases}$

В итеративной форме:

Для всех B: $\begin{cases} IN_{B,i} = \bigwedge_{\{B',B\}} OUT_{B',i-1} \\ OUT_{B,i} = T_B(IN_{B,i-1}) \end{cases}$



Поиск выхода за границы массива

Рассмотрим ошибку выход за границы массива.

Пусть происходит обращение к массиву размера N, $a[i] = b$;

Тогда, ошибка выход за границы массива произойдет в том случае, если $\exists \sigma \in P_{\{a[i]=b\}} : \sigma_i \geq N \vee \sigma_i < 0$

Посчитать $P_{\{a[i]=b\}}$ не получится, зато можно посчитать $\phi\left(l_{\{a[i]=b\}}\right) : P_{\{a[i]=b\}} \subseteq \phi\left(l_{\{a[i]=b\}}\right)$

Корректность и полнота

- Пусть происходит обращение к массиву размера N, $a[i] = b$;
- Пусть известно $\phi\left(l_{\{a[i]=b\}}\right)$

- Тогда необходимое условие ошибки формулируется как :

$$\exists \sigma \in \phi(l_{\{a[i]=b\}}) : \sigma_i \geq N \vee \sigma_i < 0$$

$$\Uparrow$$

$$\exists \sigma \in P_{\{a[i]=b\}} : \sigma_i \geq N \vee \sigma_i < 0$$

- Необходимое условие ошибки обладает свойством корректности, т.е. не пропускает ошибки.

- Достаточное условие ошибки:

$$\forall\, \sigma \in \phi(l_{\{a[i]=b\}}) : \sigma_i \geq N \vee \sigma_i < 0$$

- Достаточное условие ошибки обладает свойством полноты, т.е. не находит ложные ошибки.

0502

АБСТРАКТНАЯ ИНТЕРПРЕТАЦИЯ ПРАКТИКА

Внутреннее представление

Программа задается в виде ГПУ

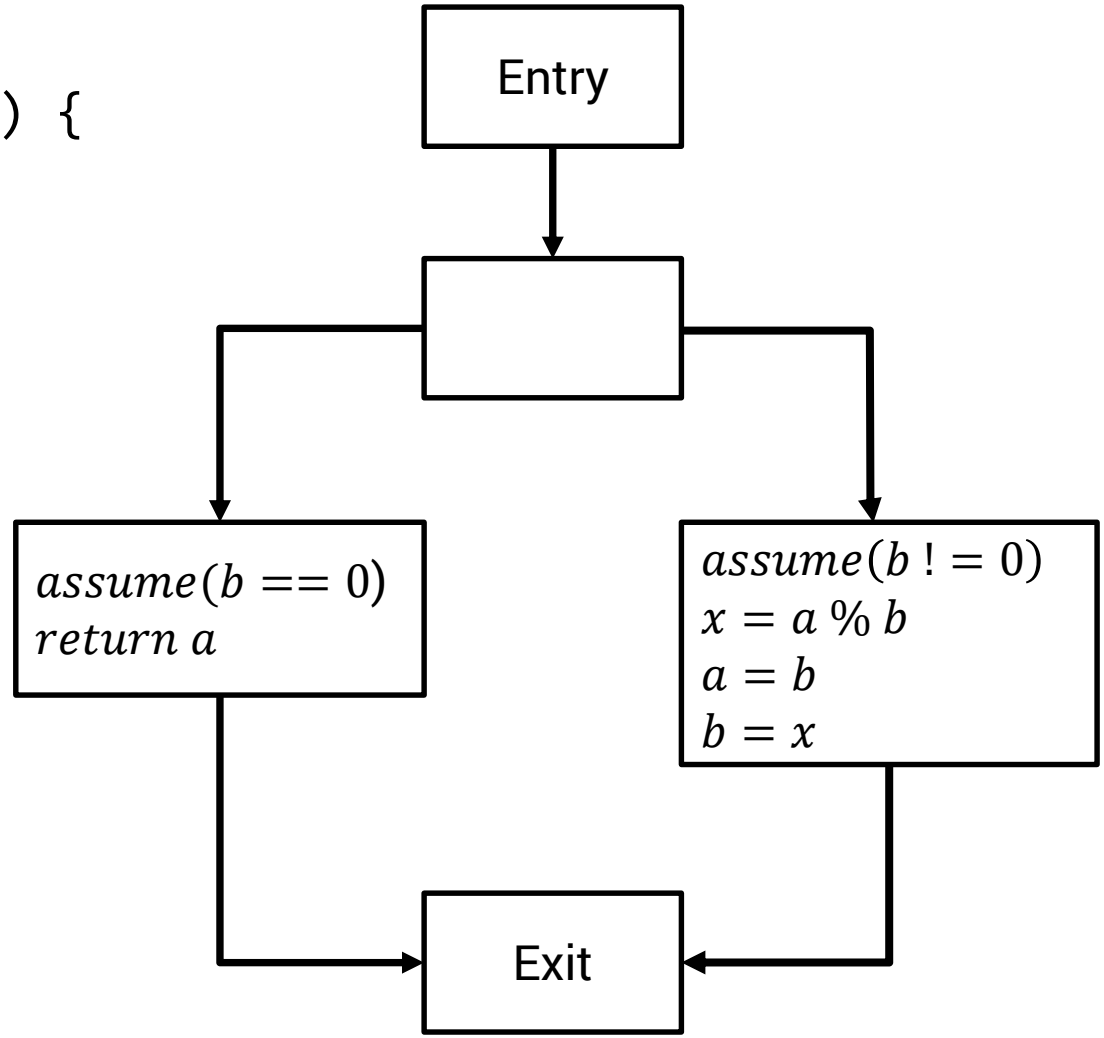
Имеется следующий набор инструкций:

- $a = b \circ c$ – бинарная арифметическая операция
- $a = \star\, b$ – унарная операция
- $a = b$ – присваивание
- $a = const$ – присваивание константы
- $assume(a \circ b\ )$ – условие перехода
- $a[i] = x$ – запись в массив
- $x = a[i]$ – чтение из массива

Инструкции assume всегда располагается в начале базового блока.

Представление программ

```
1) int gcd(int a, int b) {
2)   int x;
3)   while (b != 0) {
4)     x = a % b;
5)     a = b;
6)     b = x;
7)   }
8)   return a;
9) }
```



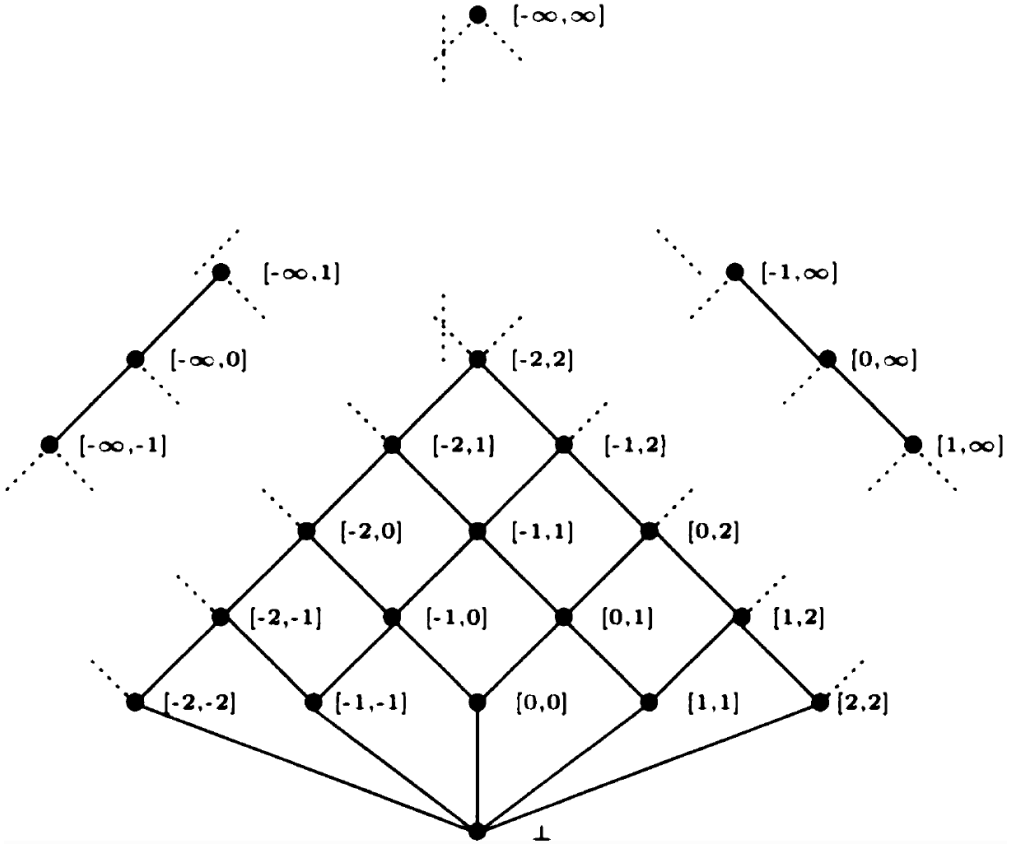
Поиск необходимых условий. Анализ интервалов значений

- Анализ интервалов значений вычисляет в каждой точке программы интервалы, покрывающие все возможные значения переменных.
- Интервалы значений позволяют обнаруживать инвариантные условия в программе.

```
if (…)
    x = 1;
else
    x = 3;

// x ∈ [1, 3]

if (x > 5) {
    ...       // unreachable code
}
```

Решётка интервалов

- Элементами решётки для одной целочисленной переменной являются интервалы **[a, b]**



- Частичный порядок определяется отношением включения ⊆

- Наименьшая верхняя граница для нескольких интервалов вычисляется при помощи объединения ∪

Вычисление интервалов значений

Передаточные функции для арифметических операций основываются на интервальной арифметике.

для переменных **x** и **y** с интервалами значений **[a, b]** и **[c, d]** и константы **k** :

Выражение	Вычисление интервала значений для результата выражения
x = k	[k, k]
x + y	[a, b] + [c, d] = [a + c, b + d]
x - y	[a, b] – [c, d] = [a – c, b – d]
x * y	[a, b] * [c, d] = [min(ac, ad, bc, bd), max(ac, ad, bc, bd)]
x / y	[a, b] / [c, d] = [min(a/c, a/d, b/c, b/d), max(a/c, a/d, b/c, b/d)]

Вычисление интервалов значений

Передаточные функции для сравнений.

для переменных **x** и **y** с интервалами значений **[a, b]** и **[c, d]** и константы **k** :

Сравнение	Вычисление интервалов значений для **x** и **y** в результате сравнения
assume x == y	x: [a, b] ∩ [c, d] y: [a, b] ∩ [c, d]
assume x ≤ y	x: [a, b] ∩ [-inf, d] y: [a, +inf] ∩ [c, d]
assume x > y	x: [a, b] ∩ [c + 1, +inf] y: [-inf, b - 1] ∩ [c, d]
assume x ≠ k	x: [a + 1, b], when k = a x: [a, b + 1], when k = b x: [a, b], when k ∈ [a + 1, b - 1] x: ∅ , when k ∉ [a, b]

Необходимость *widening*

- Поскольку полурешётка интервалов имеет бесконечную высоту, то анализ потока данных не будет сходиться.

```
// n ∈ [-inf, +inf]
for (i = 0; i < n; ++i) {
    // i ∈ [0, 0]  1-st dataflow iteration
    // i ∈ [0, 1]  2-nd dataflow iteration
    // i ∈ [0, 2]  3-rd dataflow iteration
    // ...
    // i ∈ [0, +inf]
}
```

- Чтобы анализ сошёлся, мы применяем *widening*:
  - внутри цикла на очередной итерации анализа при перевычислении интервала в точке программы:
    1) при расширении границы интервала, она сразу заменяется на inf
    2) левая граница интервала не смещается вправо, а левая не смещается влево

Пустой интервал значений

- На инструкциях сравнения может быть вычислен пустой интервал.
  - пустой интервал свидетельствует о том, что сравнение инвариантно: результат сравнения всегда ложный

```
// x ∈ [0, 1]
// y ∈ [3, 7]

if (x == y) { // assume x = y
    ...       // [0, 1] ∩ [3, 7] = ∅
    ...       // unreachable code
}
```

- Недостижимому коду соответствует значение $\bot$ из полурешётки для всех переменных.

Неучитывание выколотой точки

- Интервальная абстракция не учитывает выколотые точки.
- Из-за нехватки точности анализ не обнаруживает многие инвариантные сравнения

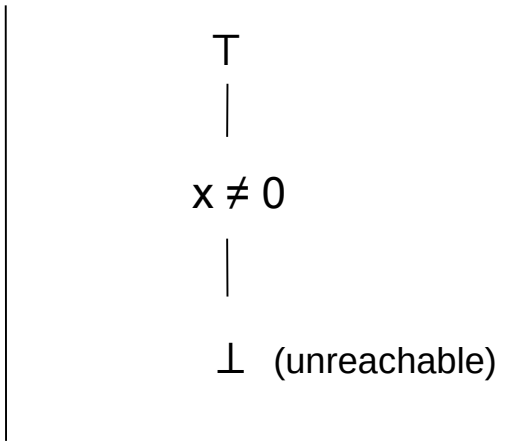
```
if (x != 0) { // assume x ≠ 0

    // ???

    if    (x == 0) { // assume x = 0

        // x ∈ [0, 0]
        // unreachable code?
        ...
    }
}
```

Анализ нулевой выколотой точки

Дополним интервальную абстракцию анализом выколотого нуля.

- Передаточные функции:

Инструкция	Значение потока данных
x = 0	⊤
x = k	x ≠ 0 , when k ≠ 0
assume x ≠ 0	x ≠ 0
assume x = 0	⊥ , если x ≠ 0



Неучитываемые сравнения

- В описанном анализе не учитываются сравнения тех переменных, которым сопоставлены значения ⊤

```
void foo(int a, int b) {

    if (a > b) {

        if (a <= b) {
            // unreachable code?
        }

    }
}
```

- Учесть подобные ситуации можно при помощи вычисления необходимых условий достижения точек программы

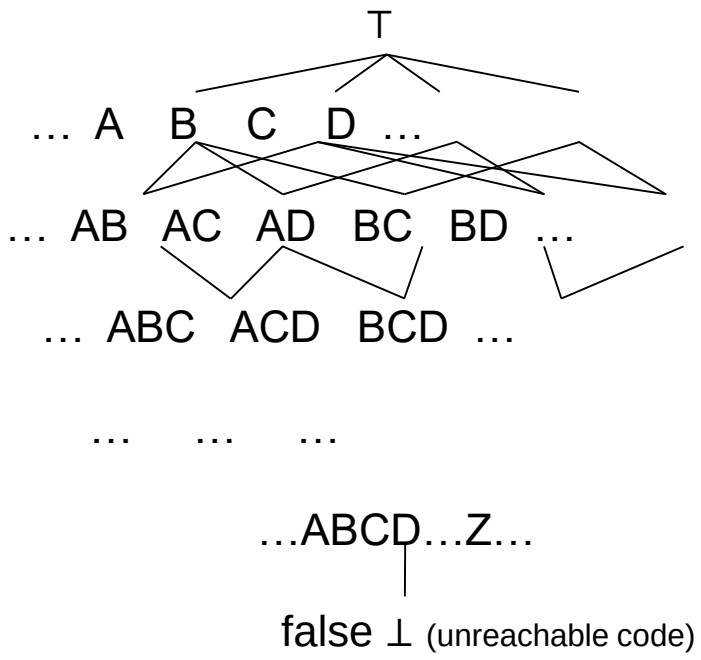
Предикатная абстракция

- Анализ производится над программой, переведённой в SSA-форму
- Предикатный анализ вычисляет необходимые условия достижения точек программы
- Вычисляемые условия – конъюнкции, состоящие из предикатов инструкций сравнения

```
1: void foo(int a1, int b1) {

2:       if (a1 > b1) { // a1 > b1

3:           if (b1 != 3) {// a1 > b1 && b1 != 3

4:           }
5:           // a1 > b1

6:       } else {
7:           // a1 <= b1

7:   }
8: }
```

Решётка предикатной абстракции

- Элементами решётки для точки программы являются конъюнкции предикатов, содержащихся в программе (предикаты обозн. A, B, C, …)
- Предикат – либо атомарное сравнение двух переменных, либо определение значения переменной.
- Каждая конъюнкция рассматривается как множество предикатов
- Частичный порядок конъюнкций определяется отношением вложенности этих множеств $\subseteq$
- Наименьшая верхняя граница для набора конъюнкций вычисляется при помощи их пересечения $\cup$



Передаточные функции предикатной абстракции

- P – предикат вида $\bigvee_i \{C_i\}$,
- $P \backslash a$ – удалить из формулы все вхождения переменной a

- $a = b \circ c$
- $a = \star\, b$
- $a = b$
- $a = const$
- $assume(a \circ b)$
- $a[i] = x$
- $x = a[i]$

- $P \rightarrow (P \backslash a) \wedge (a = b \circ c)$
- $P \rightarrow (P \backslash a) \wedge (a = \star\, b)$
- $P \rightarrow (P \backslash a) \wedge (a = b)$
- $P \rightarrow (P \backslash a) \wedge (a = const)$
- $P \rightarrow P \wedge (a \circ b)$
- $P \rightarrow P$
- $P \rightarrow P \backslash \mathrm{x}$

$P' \cap P''$ – содержит только общие коньюкты для P' и P''

Условие ошибок при предикатной абстракции

- Пусть P_B — необходимое условие для текущей точки.
- Пусть V — множество переменных в программе, $\vec{v}$ - вектор переменных. По построению P_B зависит от $\vec{v}$, т.е. $P_B(\vec{v})$
- Необходимое условие ошибки: $\exists \vec{v}: P_b\ (\vec{v}) \wedge (N \leq i \vee 0 > i)$
- Достаточное условие ошибки: $\forall \vec{v}: P_b\ (\vec{v}) \wedge (N \leq i \vee 0 > i)$
- Данные формулы можно решать при помощи SMT-солвера.

SMT-решатели

- Программы для решения уравнений в различных теориях.
- Обобщение SAT-задачи.
- Допускается использование кванторов, целочисленной арифметики, битовой арифметики и т.д.
- Стандартный язык для решателей – SMTLib2.
- Примеры решателей: Z3, CVC4

Символьное выполнение

Идея: вместо того, чтобы рассматривать все возможные пути, рассмотрим один конкретный путь.

Для выбранного пути проведем абстрактную интерпретацию

Плюсы:

Полученное необходимое условие для пути, будет точнее, т.к. нет операций сбора.

Минусы:

Нельзя перебрать все пути.

Разные чувствительности

- Чувствительность к потоку (flow sensitivity): анализ учитывает порядок инструкций
- Чувствительность к путям (path sensitivity): анализ учитывает условия переходов
- Чувствительность к контексту (context sensitivity): анализ учитывает вызовы функций