Чувствительность к пути или

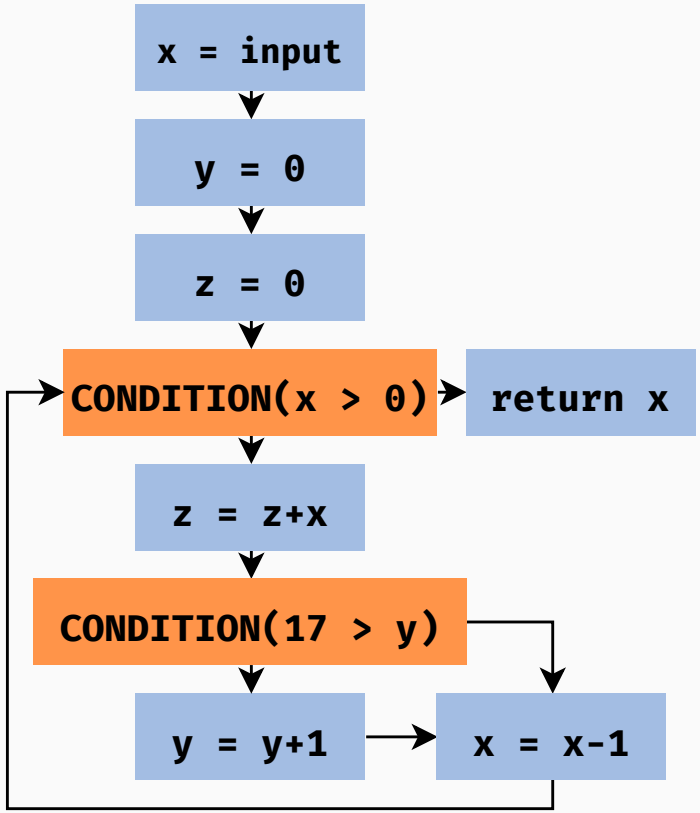
зачем в программе нужны условия

Марат Ахин Михаил Беляев

20 марта 2018 г.

Ретроспектива

Во всех рассмотренных нами анализах условия в `if`-ах не играли особой роли



Да мы их даже на картинках с CFG никак не обозначали!

Control-flow dependencies

- До этого мы рассматривали в качестве зависимостей между значениями зависимости **по данным**
- Но значения зависят ещё и от условий в переходах!
- Такие зависимости называются зависимостями **по путям** или **по управлению**

Path-sensitivity

Чувствительность к путям

Способность анализа использовать условия в переходах и циклах по-разному в разных ветках

Зачем это нужно?

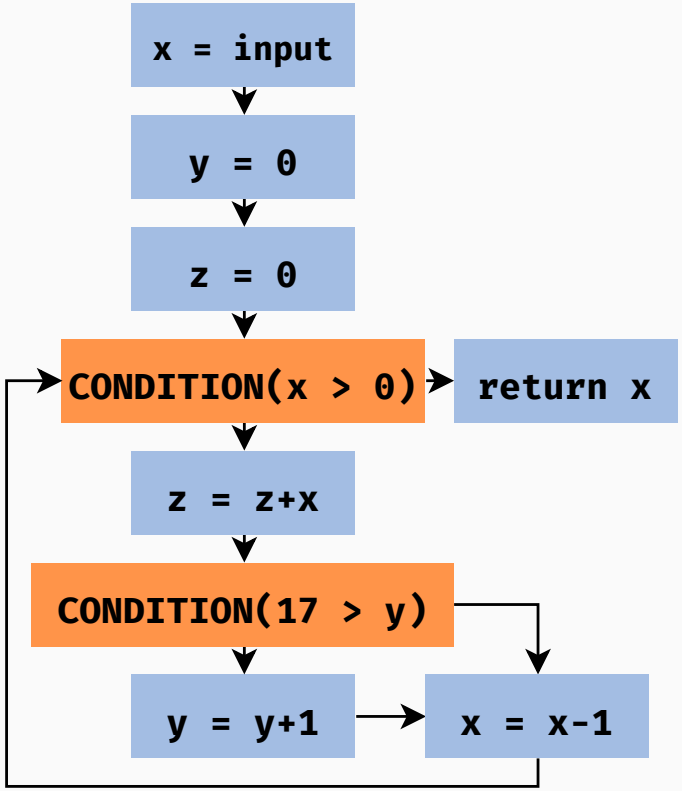
Рассмотрим пример

Даёшь интервальный анализ!

Решение в конечной точке:

$$[\![x]\!] = [-\infty, +\infty]$$
$$[\![y]\!] = [0, +\infty]$$
$$[\![z]\!] = [-\infty, +\infty]$$

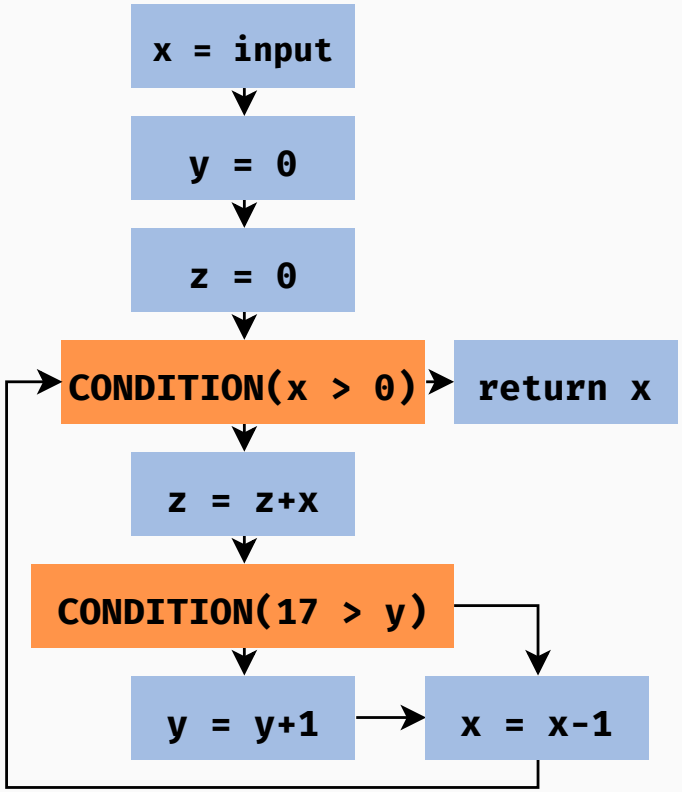
Всё нормально?



Есть вопросы

А хотелось бы так:

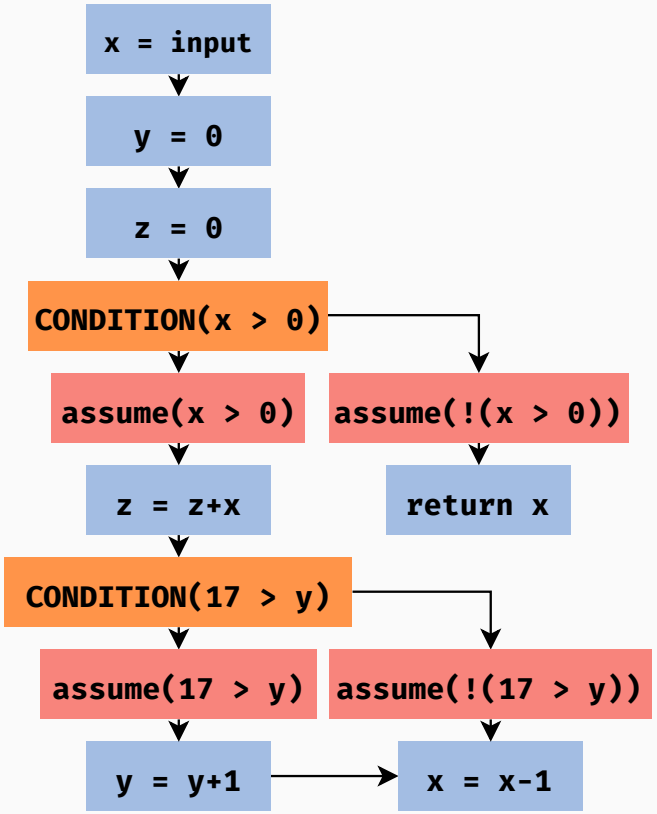
$$[\![x]\!] = [-\infty, 0]$$
$$[\![y]\!] = [0, 17]$$
$$[\![z]\!] = [0, +\infty]$$



Почему так не получается?

Предикаты путей

Чтобы не менять уже устоявшуюся нотацию, просто добавим условия в старый CFG



Что здесь значит конструкция `assume(X)`?

`assume`

- `assume` — это брат-близнец `assert`
- `assert` $\Leftrightarrow$ «проверь вот это условие здесь»
- `assume` $\Leftrightarrow$ «можешь считать, что это условие здесь истинно»

Как обрабатывать `assume/assert`?

- Для непонятных условий всегда работает стандартное

$$[\![\mathrm{n}]\!] = \mathit{JOIN}(n)$$

- Но мы-то хотели не это!

Как обрабатывать `assume/assert`?

- Для непонятных условий всегда работает стандартное

$$[\![\mathrm{n}]\!] = JOIN(n)$$

- Но мы-то хотели не это!

$$[\![\mathrm{assume}(x > E)]\!] = JOIN(n)\,[x \mapsto gt(JOIN(n)(x), eval(JOIN(n), E)))]$$

$$gt([a, b], [c, d]) = [a, b] \sqcap [c, +\infty]$$

А как насчёт других операторов?

$$eq([a,b],[c,d]) = [a,b] \sqcap [c,d]$$
$$gt([a,b],[c,d]) = [a,b] \sqcap [c,+\infty]$$
$$lt([a,b],[c,d]) = [a,b] \sqcap [-\infty,d]$$

А если переменная не слева?

А если и слева, и справа?

А с более сложными условиями?

(disclaimer: в TIP нет логических операторов)

$$[\![\text{assume}(P_1 \;\&\&\; P_2)]\!] = [\![\text{assume}(P_1)]\!] \sqcap [\![\text{assume}(P_2)]\!]$$

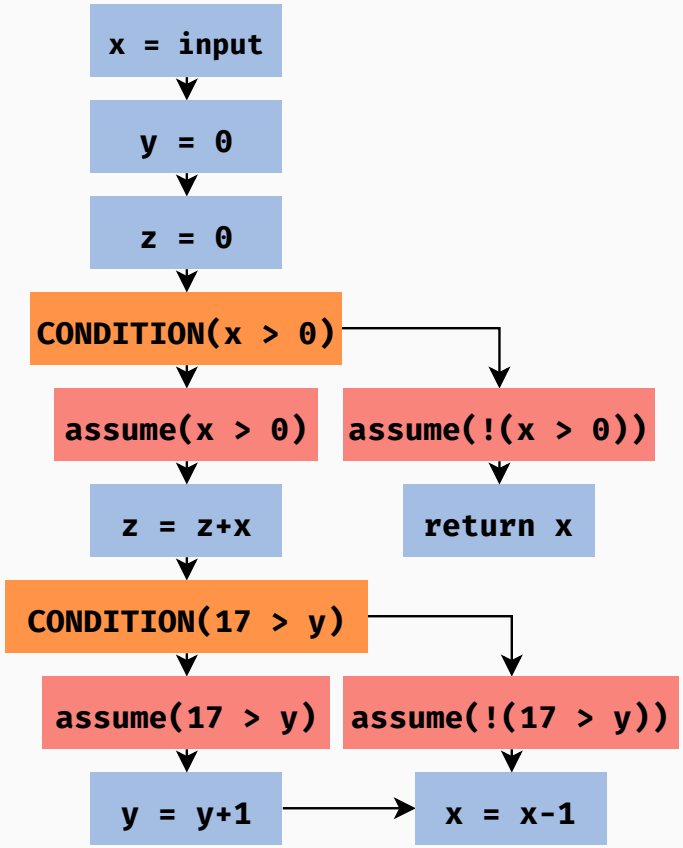
$$[\![\text{assume}(P_1 \;||\; P_2)]\!] = [\![\text{assume}(P_1)]\!] \sqcup [\![\text{assume}(P_2)]\!]$$

Итого

Получилось!

$$[\![x]\!] = [-\infty, 0]$$
$$[\![y]\!] = [0, 17]$$
$$[\![z]\!] = [0, +\infty]$$

Ура?



Проблемы

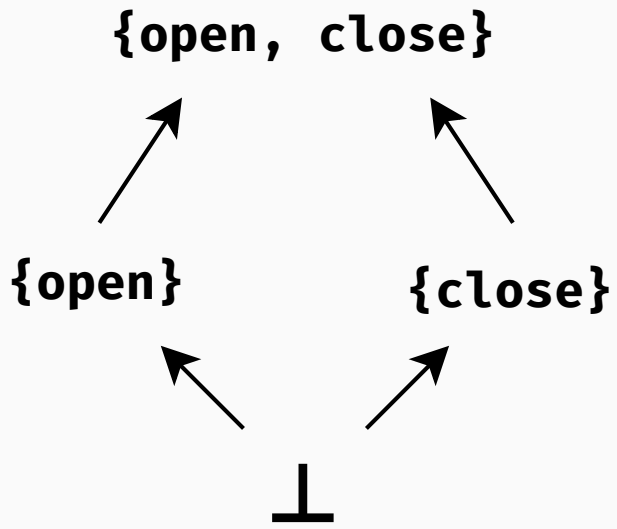
- Как быть с условиями вида `x + 1 > 4`?
  - Символически выражать через `x`
  - Иногда это сделать просто нельзя!
- Как быть с условиями вида `func(x)`?
  - Немного поговорим, когда будем обсуждать межпроцедурность =)
- Как быть с корреляцией условий?..

(Де)мотивирующий пример

Рассмотрим совсем простую решётку:

$$L = \mathcal{P}\left(\{\text{open}, \text{close}\}\right)$$

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



$$[\![\text{open}()]\!] = \{\text{open}\}$$

$$[\![\text{close}()]\!] = \{\text{close}\}$$

$$[\![\text{n}]\!] = JOIN(n)$$

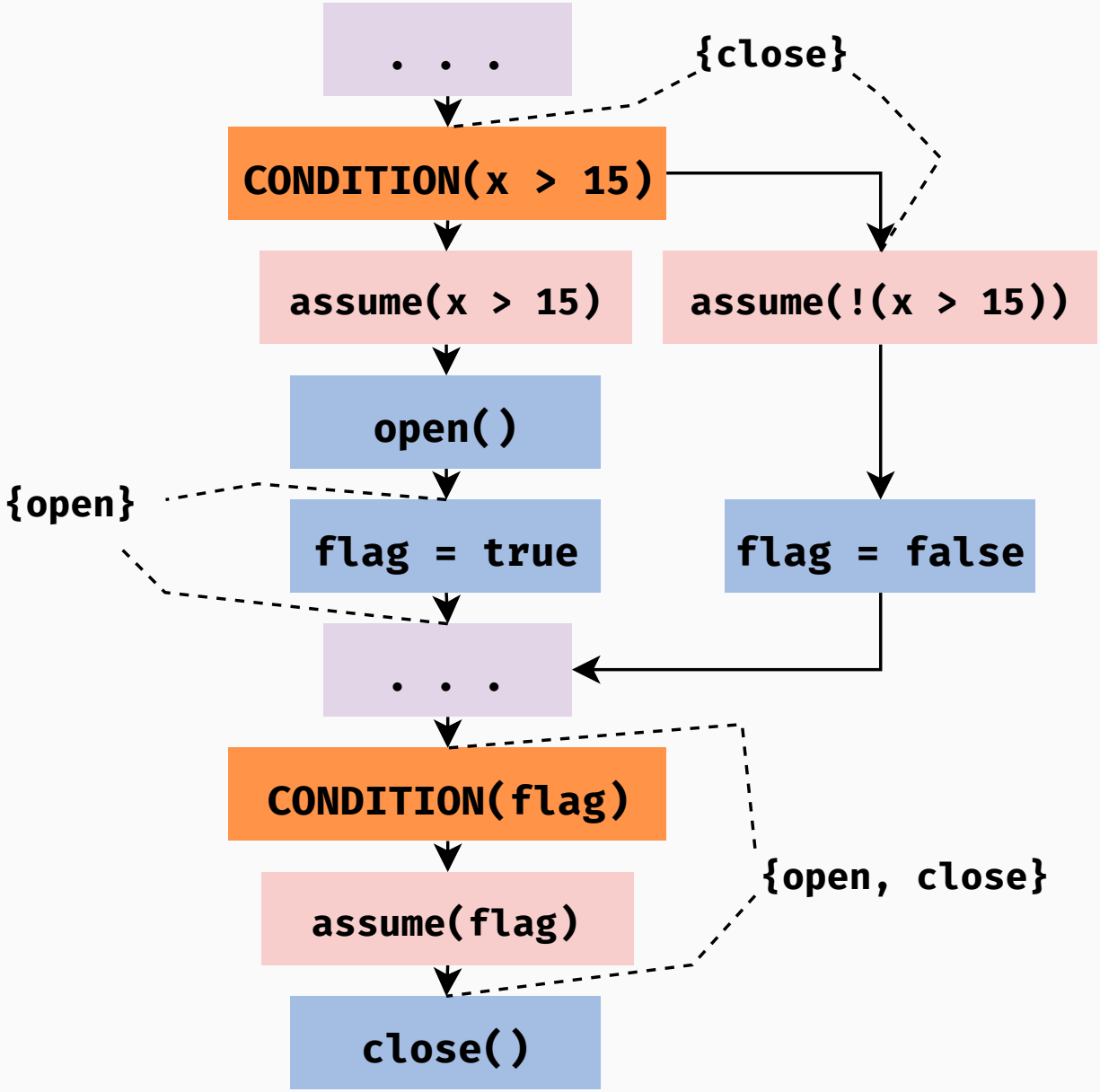
- Нельзя закрывать уже закрытый файл
- Нельзя открывать уже открытый файл
- Давайте это проверим!

(Де)мотивирующий пример

```
...
if(x > 15) {
  open();
  flag = 1;
} else {
  flag = 0;
}
...
if(flag) {
  close();
}
```

- Является ли эта программа корректной?
- А что думает наш анализ?

Наш анализ думает не то



Кто виноват и что делать?

- Виноват флаг `flag`!
- Что делать? Вносить его в анализ

Интересные решетки

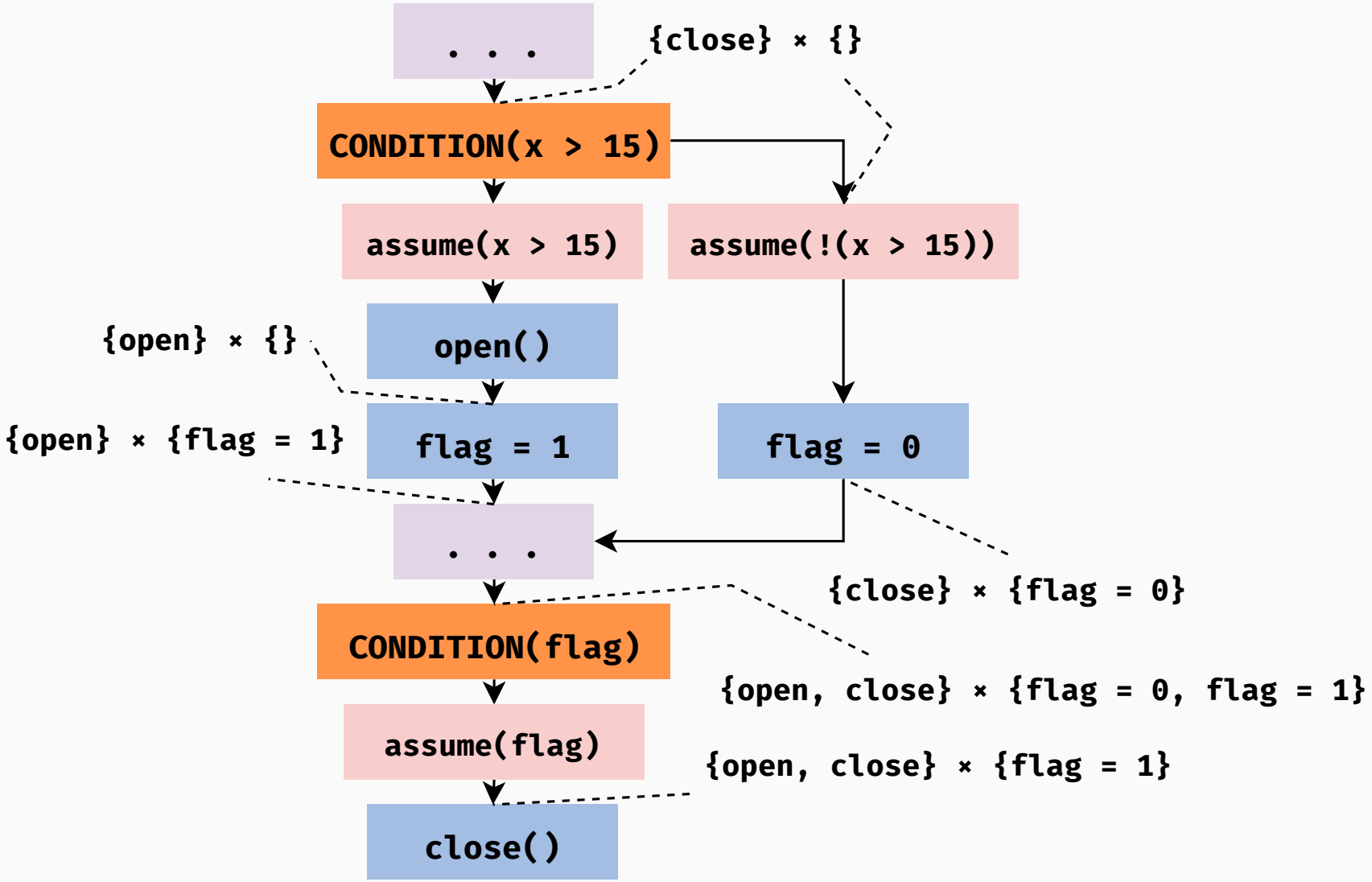
Произведение решеток

- $L_1 \times L_2 \times \ldots \times L_n = \{(x_1, x_2, \ldots, x_n) | x_i \in L_i\}$
- $(x_1, x_2, \ldots, x_n) \sqsubseteq (x'_1, x'_2, \ldots, x'_n) \Leftrightarrow \forall i : x_i \sqsubseteq x'_i$

Попытка номер 2

$$L = \mathcal{P}\left(\{\text{open}, \text{close}\}\right) \times \mathcal{P}\left(\{\texttt{flag == 0}, \texttt{flag != 0}\}\right)$$

Что получается?



Что получается?

Чего добились?

Ничего

Что делать?

Нужно как-то выразить, что `flag` и состояние зависят друг от друга

Немножко терминологии

- Множество возможных переходов называется *контекстом пути*

  В нашем случае это множество $\{\texttt{flag == 0}, \texttt{flag != 0}\}$
- Нужна решётка состояний для каждого контекста

  $L = \mathit{Paths} \to \mathcal{P}(\{\text{open}, \text{close}\})$

Есть ли с такой решёткой какие-нибудь проблемы?

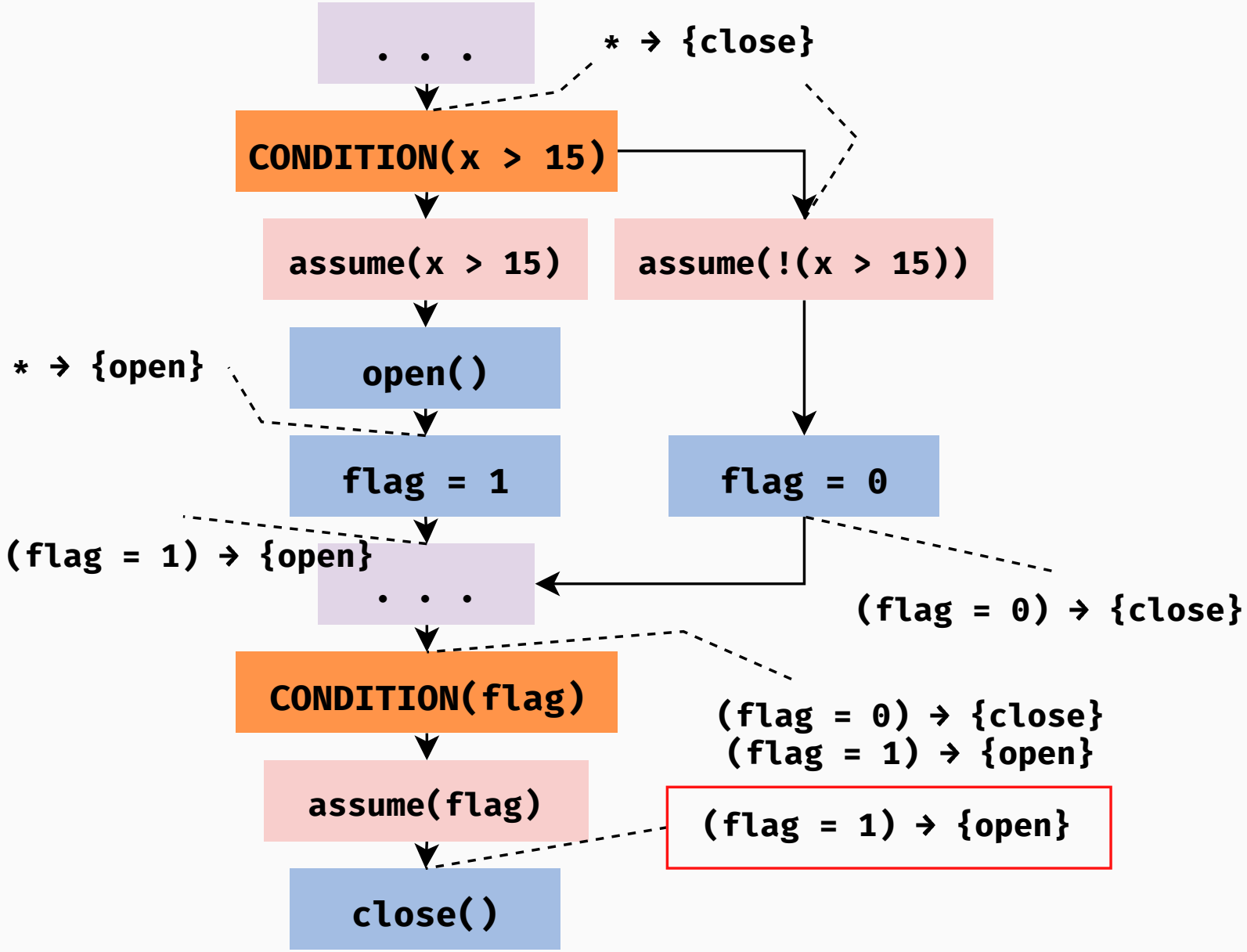
Let's eval!

$$[\![\text{assume(flag == 1)}]\!] = \mathit{JOIN}(n)\,[(\texttt{flag == 0}) \to \{\}]$$
$$[\![\text{assume(flag == 0)}]\!] = \mathit{JOIN}(n)\,[(\texttt{flag == 1}) \to \{\}]$$
$$[\![\text{flag = 1}]\!] = \mathit{JOIN}(n)\,[(\texttt{flag == 0}) \to \{\}]$$
$$[\![\text{flag = 0}]\!] = \mathit{JOIN}(n)\,[(\texttt{flag == 1}) \to \{\}]$$

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

Почему так?

Что получается?



Что получается?

- Всё отлично!
- Или нет?

Что получается?

- Всё отлично!
- Или нет?

- Непонятно, как выбирать элементы *Path*
  - Они могут зависеть ещё и друг от друга!
- Размер решётки растёт экспоненциально
- Алгоритмам от этого тоже плохеет
- Мы очень вольно определяли, что $P_1 \Leftrightarrow P_2$

  Строго говоря, это *NP*-полная задача

Как выбирать элементы *Path*

- Обычно это комбинация из условий переходов и их отрицаний
- Вплоть до $\prod_{c \in Conds} \mathcal{P}\left(\{c, !c\}\right)$, где $Conds$ — множество условий
- Полная комбинация — это $2^{|Conds|+1}$

CEGAR

Counterexample-guided abstraction refinement (BLAST-style)

1. Запускаем анализ с какой-то абстракцией
2. Если всё хорошо — закончили
3. Если есть нарушение — генерируем **абстрактный контрпример**
    - Т.е. путь, на котором это происходит
4. *Пытаемся найти конкретный контрпример*
    - Т.е. реальные значения переменных с таким же путём
5. Если он есть — закончили
6. Используем контрпример для **улучшения** абстракции
7. Назад к пункту 1

Как искать конкретный контрпример?

- Это NP-полная или около того задача
- С использованием BDD или SMT-солверов
- Этому можно посвятить отдельную лекцию

Как использовать контрпример для улучшения абстракции?

- Нужно найти неверные факты в контрпримере
- Добавить переменные, от которых они зависят

Что ещё можно делать?

- Нормализовать предикаты
- Разбить переменные на кластеры
- Сначала запустить вспомогательные анализы
  - Constant propagation
  - Dead code analysis
- Dead code propagation

$$[\![\text{open}()]\!] = \lambda p.\, \{open\}$$

$$\Downarrow$$

$$[\![\text{open}()]\!] = \lambda p.\, if(JOIN(n)(p) = \{\})\, \{\}\, else\, \{open\}$$

В следующей серии

Как во все эти простые, эффективные и работающие на практике алгоритмы вписывается наличие в программе нескольких функций?

akhin@kspt.icc.spbstu.ru

belyaev@kspt.icc.spbstu.ru