

Оптимизация строковой решетки для статического анализа кода на базе абстрактной интерпретации

Студент: Можаяев Андрей Михайлович

Руководитель: Кичин Егор Андреевич

Университет ИТМО

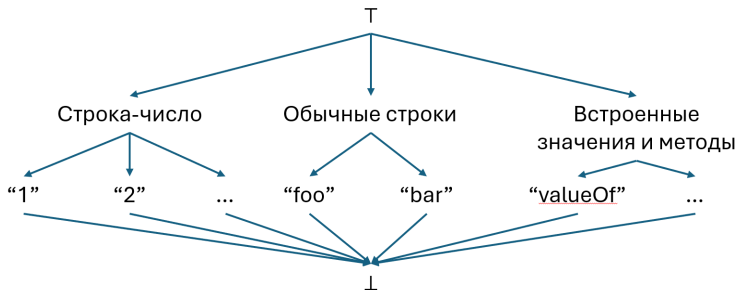
2025

Предметная область

- **Абстрактная интерпретация** – метод статического анализа, который аппроксимирует поведение программы, работая с абстрактными значениями вместо конкретных. Позволяет анализировать все возможные пути выполнения без запуска программы
- **Решетки** – это частично упорядоченное множество, обобщающее возможные значения переменной. Они используются для перехода между абстрактными значениями

Текущая строковая решетка

Точность абстракции низкая – `str.length` невозможно оценить, т.к. знаем только то, что `str` – обычная строка. Поэтому интерпретатор видит деление на 0 там, где его нет



```
string str;  
if (b) {  
    str = "foo";  
} else {  
    str = "bar";  
}  
return 100 / str.length;
```

Цель и задачи

Цель: повысить точность строковой решетки, используемой в статическом анализаторе кода на базе абстрактной интерпретации

Задачи:

1. Провести обзор существующих строковых абстрактных решеток
2. Спроектировать структуру более точной строковой решетки на их основе
3. Реализовать операции для работы с абстрактными значениями строк и интегрировать в статический анализатор
4. Протестировать новую функциональность, провести замеры производительности статического анализатора при использовании новой решетки

Существующие решения

Домен	Операции	Работа с T	Скорость
Abstract Length	length	Минимальная длина	Высокая
Character Inclusion	contains	Только символы	Высокая
Prefix-Suffix	starts/endsWith substring	Префиксы суффиксы	Высокая
String Set (SS_k)	Все	Переполнение в T	Средняя
Конечные автоматы	Все	Через T -переходы	Очень низкая
TARSIS ¹	Все (с ограничениями)	Через T -переходы	Средняя

За основу был выбран подход TARSIS, так как обладает хорошей точностью при средней производительности

¹ Arceri V. et al. TARSIS: Abstract Domains for String Analysis (2022) DOI: 10.48550/arXiv.2006.02715

Структура решетки

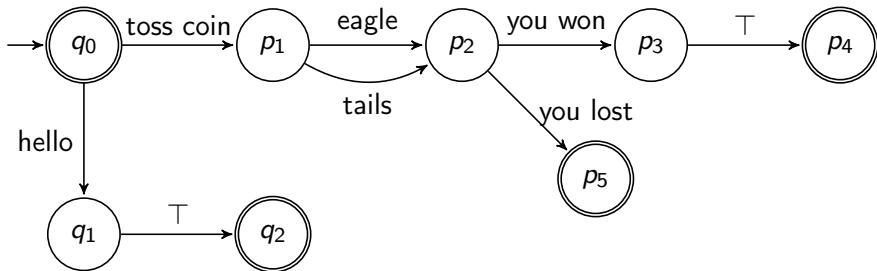
Множество значений строк можно представить в виде автомата. На автоматах задан порядок по включению, есть операции объединения и пересечения

$$\langle \mathcal{TF}_A, \sqsubseteq_{\mathcal{T}}, \sqcup_{\mathcal{T}}, \sqcap_{\mathcal{T}}, \text{Min}(\emptyset), \text{Min}(\mathbb{A}^*) \rangle$$

- $\mathbb{A}^* = \Sigma^* \cup \{\top\}$ – Алфавит, состоящий из слов и специального символа \top , означающего любую строку
- \mathcal{TF}_A – Множество автоматов над алфавитом \mathbb{A}^*
- $\sqsubseteq_{\mathcal{T}}$ – Частичный порядок на автоматах, основанный на включении языков
- $\sqcup_{\mathcal{T}}$ и $\sqcap_{\mathcal{T}}$ – Наибольший (join) и наименьший (meet) общий по включению элемент в решетке
- $\text{Min}(\emptyset)$ – Автомат, принимающий пустой язык, или \perp
- $\text{Min}(\mathbb{A}^*)$ – Автомат, принимающий любой язык, или \top

Пример

```
str = ""  
if (play) {  
    coin = tossCoin()  
    str = "toss coin" + coin + calcBet(coin)  
} else if (greetings) str = "hello " + name()
```



Реализованные операции

- Операции для работы с автоматами
 - Для решеток – **join** ($A \cup B$), **meet** ($A \cap B$), **weakerThen** ($A \subset B$)
 - Вспомогательные – **minimize**, **explode**, **collapse**, **getLanguage**
- Строковые операции. Работая с автоматами, то есть множеством возможных строк, определяем множество возможных значений при применении операции
 - TARSIS – **length**, **concat**, **contains**, **substr**, **replace**, **indexOf**
 - Дополнительно реализованные – **charAt**, **starts/endsWith**, **trim**, **pad**, **toBool**, **toNum**, **strLt**
- Так как решетка содержит бесконечную цепочку вложенных автоматов, то для нахождения неподвижной точки при анализе циклов реализован оператор расширения – **widening**

Новая функциональность

```
function tossCoin(): string {  
  if (Math.Random() > 0.5) {  
    return "eagle"  
  } else {  
    return "tails"  
  }  
}
```

```
let x = tossCoin()           // Было -> Стало  
console.log(x.indexOf("a")) // T    -> [number: [1, 1]]  
console.log(x.indexOf("e")) // T    -> [number: [-1, 4]]  
console.log(x.indexOf("r")) // T    -> [number: [-1, -1]]
```

Новая функциональность

```
var x = "Hello " + name()           // Было -> Стало
console.log(x)                       // T    -> [string: Hello T]
console.log(x.length)                // T    -> [number: [6, +Inf]]
console.log(x.startsWith("He"))      // T    -> [boolean: true]
console.log(x.substr(0, 4))           // T    -> [string: Hell]
```

Сравнение точности

Сравниваем количество возвращаемых значений функций при анализе

Операция	Значение	Было	Стало	Изменение
Concat	Top	319,381	25,331	↓ 92%
	Const	58,321	58,767	↑ 1%
	Automaton	0	298,822	-
Возвращает String	Top	4,587	3,610	↓ 22%
	Const	795	1,363	↑ 71%
	Automaton	0	201	-
Возвращает Number	Top	823	426	↓ 48%
	NonTop	477	801	↑ 68%
Возвращает Boolean	Top	15,470	14,326	↓ 7%
	NonTop	199	287	↑ 44%

В среднем точность строковых операций возрасла на 67%

Сравнение производительности

Сравниваем время анализа, количество проанализированных состояний программы и среднее потребление памяти всего анализатора

Тип	Метрика	Было	Стало	Изменение
Суммарно	Время (мин)	28:25	33:23	↑ 18%
	Состояния	35.7 млн	35.3 млн	↓ 1%
Малые до 30 сек	Время	0:22	0:22	↑ 0%
	Память	4к МБ	4.3к МБ	↑ 7%
Средние 0.5-5 мин	Время	1:19	1:38	↑ 24%
	Память	7.1к МБ	6.8к МБ	↓ 4%
Большие от 5 мин	Время	8:26	8:46	↑ 4%
	Память	17.9к МБ	16.7к МБ	↓ 7%

Сокращение памяти обусловлено тем, что меньше состояний программы сохраняется

Open Source

Операция	Метрика	Было	Стало	Изменение
Суммарно	Время (мин)	2:39	3:12	↑ 20%
	Состояния	5.47 млн	5.29 млн	↓ 3%
Concat	Top	17,457	3,908	↓ 78%
	Const	2038	2029	↓ 1%
	Automaton	0	13,247	-
Возвращает String	Top	130	117	↓ 10%
	Const	55	62	↑ 13%
	Automaton	0	12	-
Возвращает Number	Top	24	5	↓ 80%
	NonTop	3	22	↑ 633%
Возвращает Boolean	Top	3090	3010	↓ 3%
	NonTop	3	8	↑ 167%

Результаты

- На основе обзора строковых абстрактных решеток выбран подход TARSIS
- На базе его спроектирована структура строковой решетки
- Разработанная решетка реализована и интегрирована в статический анализатор кода.
- Удалось повысить точность анализа строковых операций на 67% в среднем
- Суммарное время исполнения увеличилось не более, чем на 20%

Дополнительные материалы

Сужение

```
var text = read(filename)
while (text.contains("chapter")) {
    var fromIdx = text.indexOf("chapter") // [number: [-1, +Inf]]
    // Interpreter indicates a potential error here: substr(-1)
    var curChapter = text.substr(fromIdx)
    ...
}
```

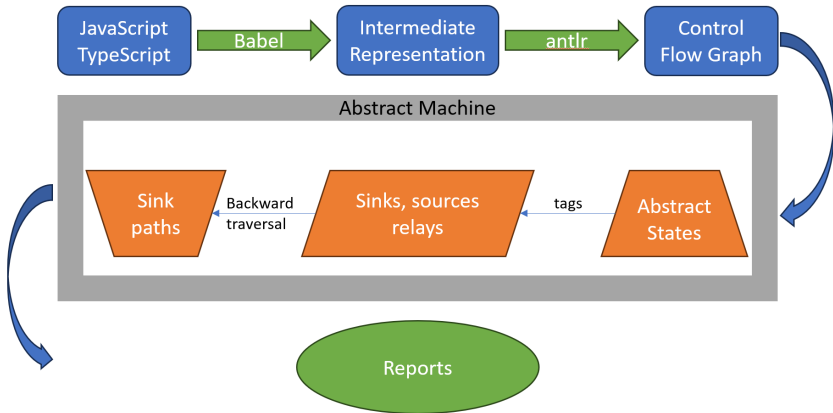
В планах реализация сужения (narrowing), то есть получения дополнительной информации о строке при заходе в ветку

Переполнение

```
var d = dayOfWeek(date)
var m = month(date)
var y = year(date)
var str = d + m + y
if (str.contains("friday december") {
    ...
}
```

Некоторые операции требуют приведения к автомату с переходами по символам вместо слов. Это может сильно замедлить анализ, поэтому в такие моменты происходит переполнение в \top

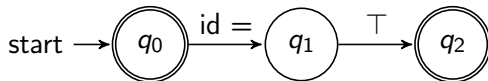
Анализатор



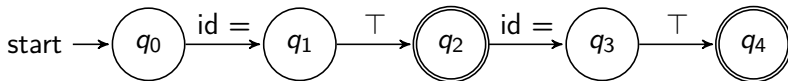
widening

```
res = "";  
while(?) res = res + "id = " + f();  
return res;
```

Начало второй итерации:

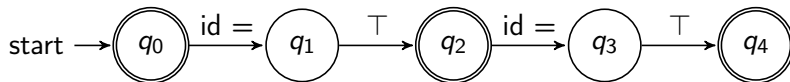


Конец второй итерации:

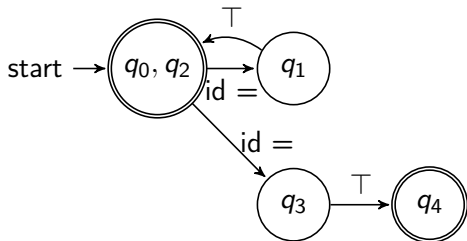


widening

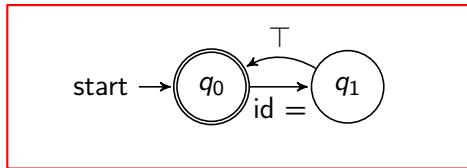
Объединенное состояние после второй итерации:



widening — Склеивание вершин с одинаковыми исходящими путями длины 2:



minimize:



Операции

$$\text{contains}(s, s') \triangleq \begin{cases} \text{false} & \text{if } A' \sqcap \text{FA}(A) = \emptyset \\ \text{true} & \text{if } \neg \text{cyclic}(A) \wedge \text{singlePath}(A') \\ & \wedge \forall \pi \in \text{paths}(A). \sigma_{sp} \curvearrowright_s \sigma_{\pi} \\ \{\text{true}, \text{false}\} & \text{otherwise} \end{cases}$$

- $\text{FA}(A)$ - Автомат, принимающий все подстроки A . Для этого все вершины A помечаем терминальными
- singlePath - Для любых двух слов, принимаемых автоматом, одно является префиксом другого. Или же автомат выглядит как путь, у которого некоторые вершины на пути помечены терминальными