DOI: https://doi.org/10.36910/6775-2524-0560-2023-50-08
УДК 004.02
Касянчук Дмитро Павлович, магістрант,
https://orcid.org/0009-0004-2824-8232
Марченко Олександр Іванович, к.т.н., доцент,
https://orcid.org/0000-0002-4537-3420
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», м. Київ, Україна

МОДИФІКОВАНИЙ МЕТОД СТАТИЧНОГО АНАЛІЗУ КОДУ ДЛЯ РІШЕННЯ ЗАДАЧІ ЗГОРТКИ РЯДКОВИХ КОНСТАНТ

Касянчук Д.П., Марченко О.І. Модифікований метод статичного аналізу коду для рішення задачі згортки рядкових констант. У даній статі запропонований модифікований метод згортки рядкових констант, який має меншу складність і вищу точність, ніж існуючі методи. В основі цього методу лежить анотування пропагованих значень додатковою інформацією, аналіз якої дозволяє не здійснювати конкатенацію значень, які не можуть існувати разом під час реального виконання програми, що, в свою чергу, дозволяє зменшити розмір проміжних результатів аналізу потоку даних, тим самим покращуючи його швидкодію і точність.
Ключові слова: граф потоку виконання, CFG, аналіз потоку даних, DFA, MFP алгоритм, MOP алгоритм, повна решітка, абстрактний домен, статичний аналіз, згортка констант.

Kasianchuk D.P., Marchenko O.I. Modified method of static code analysis for solving problem of folding of string constants. This article proposes a modified method for solving problem of propagation of string constants with less complexity and higher accuracy than existing methods. The basis of the considered method is the annotation of the propagated values with additional information, the analysis of which allows not to perform the concatenation of values which cannot exist together during the actual execution of the program. This approach allows to reduce the size of the intermediate results of data flow analysis, thereby improving its speed and accuracy.
Keywords: control flow graph, CFG, data flow analysis, DFA, MFP algorithm, MOP algorithm, full lattice, abstract domain, static analysis, constant propagation.

Постановка наукової проблеми.

Рядки широко використовуються в сучасних мовах програмування, зокрема у рядкових літералах для надання інформації користувачу, у програмах обробки текстів, у програмах, що використовують рефлексію тощо. Наприклад, у мові PHP рядки є способом обміну між програмами, тоді як у JAVA вони широко використовуються у якості SQL запитів, або для отримання доступу до інформації про класи через рефлексію. Під час виконання інструкції str.substring(str.indexOf('a')) виникне помилка у разі, якщо рядок не містить такого символу. Тобто, у цьому випадку було б корисно мати можливість відслідковувати множину можливих символів, які належать конкретному рядку. Іншим же прикладом є робота з SQL запитами. Наприклад, після виконання такого запиту – "delete from some_tbl where id = " + id, де id – "10 OR TRUE" – весь зміст таблиці буде втрачений без можливості відновлення. Таким чином, неправильні маніпуляції з рядками можуть призвести не тільки до виняткових, а також до драматичних і невідновлюваних ситуацій.

З вищесказаного можна зробити висновок, що важливість методів, які автоматично аналізують і виявляють помилки, що можуть виникнути при роботі з рядками, є дуже високою. Але існуючі методи такого типу все ще не повністю задовольняють сучасним вимогам: методи, які базуються на використанні автоматів та регулярних виразах є точними, але повільними, і, головне їх швидкодію важко покращити; багато інших методів є доволі швидкими, але вони є досить спеціалізованими, оскільки орієнтовані на певні властивості або класи програм. Тому задача розробки нових методів, які не були б вузько спеціалізованими і достатньо швидкими, є актуальною.

Перелік скорочень, умовних позначень та термінів.

CFG – граф потоку виконання (англ. – Control Flow Graph).
DFA – аналіз потоку даних (англ. – Data Flow Analysis).
MFP – максимальна фіксована точка (англ. - Maximal Fixed Point).
MOP – об'єднання всіх шляхів (англ. - Meet Over All Paths).
Трансферна функція – операція DFA, мета якої модифікація вхідних властивостей вузла графу після аналізу інструкцій у ньому.
Join-функція – операція DFA, мета якої обробка декількох можливих результатів і отримання на їх основі єдиного.

Аналіз досліджень.

Як було сказано вище, на сьогоднішній день існує невелика кількість методів та інструментів, які здатні ефективно отримати множину константних значень рядкових змінних у певних локаціях цієї програми. Розглянемо методи та підходи для задачі згортки констант, що описані в [1,2].

Методи, що запропоновані в [1], орієнтовані на знаходження таких значень змінних програми, які є сталими для змінної протягом виконання всієї програми, тобто, фактично, такі змінні є константами у цій програмі. Іншими словами, розмір множини можливих значень цієї змінної не перевищує одиниці. Незмінність значення змінної протягом програми контролюється join-функцією. Значення, яке вона повертає обчислюється наступним чином:

$$\begin{cases} x_1, if\ x_1 = x_2 \\ \top, if\ x_1 \neq x_2\ or\ x_1 = \top\ or\ x_2 = \top \end{cases}$$

де x_1, x_2 – значення змінної, що були отримані при аналізі попередніх інструкцій,
$\top$ – спеціальний символ для позначення невідомого значення.

Методи, що описані в [2], створенні конкретно для задачі аналізу рядкових констант і базуються

на визначенні абстрактного домену, а також його властивостей. Підсумовуючи, можна виділити наступні домени:

1. $\overline{CI}$ домен, метою якого є апроксимування рядків до двох множин символів: які точно або можливо в ньому містяться. Дана інформація є корисною для відстеження доступу до індексів через пошук певного символу у рядках. Результатом join-функції є перетин множин точних символів і об'єднання множин можливих символів.
2. $\overline{PR}$ та $\overline{SU}$ домени, метою яких є апроксимування рядків до префіксів і/або суфіксів. Результатами join-функції є найбільший спільний префікс і суфікс рядків відповідно. Дані домени можуть бути використанні для перевірки деяких простих синтаксичних властивостей (наприклад, перевірка рядка, який використовується у якості SQL команди, на наявність «SELECT» слова на початку і «;» в кінці) при всіх можливих потоках виконання.
3. В домені $\overline{SG}$ для представлення множини можливих рядків використовується граф. Вузли графа описують або рядки, або операцію альтернативи (OR-вузол), або операція над рядками (в даному випадку – це функція об'єднання). Join-функція продукує новий граф, який об'єднує вхідні значення OR-вузлом. Результат роботи даного аналізу може використовуватися у інструментах, які потребують повну множину можливих рядків без обмежень.

Неважко помітити, що метод на основі виділення домену $\overline{SG}$ є більш точним, ніж на основі $\overline{CI}, \overline{PR}, \overline{SU}$, які мають не дуже велику точність, але алгоритми яких зберігають лінійну складність, в той час як алгоритм методу на основі $\overline{SG}$ домену є більш складним (на практиці складність є поліноміальною), але в той же час він набагато точніший.

Підсумовуючи, відзначимо, що описані вище методи мають наступні недоліки:

1) направленість на вузькі домени;
2) можливий комбінаторний вибух [5] складності аналізу при великій кількості розгалужень.

Постановка завдання.

На основі вищенаведеного аналізу можна зробити висновок про доцільність створення методу, який буде таким же точним, що й метод на основі $\overline{SG}$ домену, але матиме меншу складність, що дозволить використовувати його у більшій кількості ситуацій, в яких потрібно визначати повну множину рядків, що використовуються в програмі, але наявні методи не можуть бути ефективно використані через їх незадовільну швидкодію.

Завданням даної статті є розробка нового, достатньо точного та швидкого методу для вирішення задачі згортки рядкових констант на основі ідей, розглянутих у аналізі.

Запропонований метод згортки рядкових констант.

Для вирішення задачі згортки рядкових констант в якості базового був обраний підхід з використанням методів статичного аналізу коду, до яких належать і розглянуті вище у аналізі методи. На рис.1 зображено показано загальну схему статичного аналізу коду.

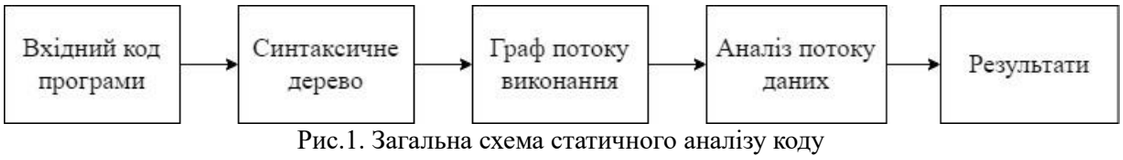

Рис.1. Загальна схема статичного аналізу коду

Зазвичай, задачі першого етапу, а саме перетворення текстового представлення програми у синтаксичне дерево, відповідно до певної граматики, досягається за допомогою автоматичних генераторів [3] синтаксичних аналізаторів.

Синтаксичні дерева забезпечують «природнє» представлення граматичної структури коду, але вони не дуже добре відповідають представленню інших властивостей програм. Тому, для моделювання різних аспектів поведінки програм в методах статичного аналізу коду використовуються більш загальні графи іншого виду.

Для виконання статичного аналізу коду з метою згортки рядкових констант автори пропонують використовувати CFG [4]. Цей граф представляє програму у вигляді окремих базових блоків коду програми та шляхів виконання цих блоків. Базовий блок коду – це блок послідовних операторів програми, в яких немає розгалужень чи циклів. CFG широко використовується для моделювання поведінки програм. Вузли цього графу містять інформацію про базові блоки коду програми, а ребра – про передачу керування між ними.

На основі CFG виконується DFA. Суть цього аналізу полягає у анотуванні кожного базового блоку у вузлах графу певними властивостями. Як правило, для вирішення задачі згортки констант (рядкових і числових), з метою зберігання різних властивостей базових блоків у вузлах графу, використовується хеш-таблиця, де ключем хешування є змінна, а значенням – множина можливих констант, які ця змінна може набувати до моменту виконання того базового блоку, що представляє даний вузол.

Для поширення (передавання) значень між базовими блоками, інформація про які міститься у вузлах графу, використовується ітеративний підхід. У початковій конфігурації інформація про базові блоки у вузлах ініціалізується початковими значеннями, наприклад, пустою хеш-таблицею. Потім відбувається поширення значень змінних базових блоків по вузлах CFG вздовж його ребер. Трансферна функція позначається, як $f_{l \to l'}$,

де l – вузол, в якому відбувається зміна властивостей,

l' – вузол, якому приходять нові властивості.

Для реалізації такого ітеративного підходу був обраний MFP алгоритм, в якому властивості в блоці l' визначаються наступним чином[6]:

$$propertyAtNode(l') = \cup_{l \in pred(l')} f_{l \to l'}(propertyAtNode(l)),$$

де $pred(l')$ - це множина всіх вузлів CFG, з яких можна потрапити у вузол l'. Аналіз продовжується поки значення властивостей базових блоків у вузлах графу не стабілізуються (тобто перестануть змінюватися після чергової ітерації). Псевдокод цього алгоритму показаний на рис.2.

```
Вхідні дані:
 - Граф потоку виконання (CFG)
 - Трансферна функція
 - Початкові дані для кожного вузла
Вихідні дані:
 - Інформація про потік даних для кожного вузла

// Крок 1: Ініціалізація
for each basic block B in CFG {
    in[B] = початкове значення
    out[B] = початкове значення
}
worklist = множина всіх вузлів графу

// Крок 2: Поширення значень
while worklist is not empty {
    B = обраний вузол з worklist
    for each predecessor of B {
        in[B] = in[B] ∪ out[P] // об'єднання властивостей з батьківських вузлів
    }
    out[B] = transfer(B, in[B]) // обробка вхідних властивостей трансферною функцією
    if out[B] has changed {
        for each successor of B {
            add S to worklist
        }
    }
    remove B from worklist
}
```

Рис.2. Псевдокод MFP алгоритму

Варто зазначити, що для рішення цієї ж задачі використовується також алгоритм MOP [6]. Його відмінність полягає у тому, що значення у вузлах вираховуються для кожного шляху окремо і наприкінці обчислень об'єднуються. Досить очевидно, що результат, отриманий за допомогою такого алгоритму, буде значно точнішим, ніж результат MFP алгоритму, але швидкодія MOP алгоритму буде значно гіршою.

Розглянемо простий, але в той же час наочний, приклад типової програми, псевдокод якої показаний на рис.3. В даній програмі відбувається формування SQL запиту шляхом об'єднання значень змінних query, where і order. Зазначимо, що змінні where і order відповідають SQL операторам фільтрації і вибірки відповідно, і формуються по-різному, в залежності від параметру flag. Цей приклад показує причину збільшення розміру множини рядків, які можуть бути присвоєні певній змінній базового блоку у вершині графу, при використанні трансферної функції, яка базується на декартовому добутку, навіть для невеликої програми.

```
query = "SELECT id1, id2 FROM tbl ";

if (flag == 1) {
  where = "WHERE id1 iS NOT NULL ";
  order = "ORDER BY id1";
} else if (flag == 2) {
  where = "WHERE id2 IS NOT NULL ";
  order = "ORDER BY id2";
} else {
    where = "";
    order = "";
}

query = query + where + order;
```

Рис.3. Приклад програми з декількома розгалуженнями

Для прикладу, припустимо, що ліва частина операції присвоювання представлена рядковою змінною, в той час, як права частина – це її нове значення, яке складається зі списку впорядкованих сутностей, що об'єднуються в один рядок. Сутність, в даному випадку, може бути змінною або рядковим літералом.

Опишемо кроки знаходження нового стану змінної у вузлі графу.

1. У випадку, якщо сутність – це змінна, то визначається множина можливих значень, які вона може набувати. Дана множина міститься в таблиці стану змінних, яка була отримана при аналізі попередніх вузлів. Якщо сутність представлена рядковим літералом, то

множина складається тільки з цього літералу.

2. Обчислюється декартів добуток множин, які були отримані на попередньому кроці.
3. Складові елементів декартового добутку об'єднуються між собою.
4. Новий стан змінної, який представлений множиною, що була отримана на попередніх кроках, передається у вихідні властивості вузла (тобто на вхід наступним вузлам).

Псевдокод трансферної функції, кроки якої описані вище, зображено на рис.4.

```
function transfer (node, dfaIn) {
    stmt = getNodeStmt(node)
    if (getStmtOperation(stmt) === 'assign') {
        to = getAssignTarget(stmt)
        newValues = getAssignValues(stmt)
                    .map(v => isVariable(v) ? dfaIn[v] : [v])
                    .cartesian()
                    .map(values => values.join())
        dfaIn[to] = new Set(newValues)
    }
    return dfaIn
}
```

Рис.4. Псевдокод трансферної функції

Псевдокод join-функції зображений на рис.5.

```
function join (in1, in2) {
    return in1.mergeWith(in2, (set1, set2) => Set.union(set1, set2))
}
```

Рис.5. Псевдокод join-функції

На рис.6 показано результат DFA, обчислений стандартним методом. Вузли графу представлені еліпсами, нарисованими суцільними лініями, передача управління представлена ребрами, нарисованими суцільними лініями. Ребра, які нарисовані пунктиром, з'єднують вузли графу і блоки з пунктирними рамками. Ці блоки описують стан змінних, який буде отриманий після аналізу інструкції в даному блоці за допомогою трансферної функції. Крім того, граф містить спеціальний вузол JOIN, який показаний пунктирною рамкою і був доданий для позначення місця виконання join-функції.

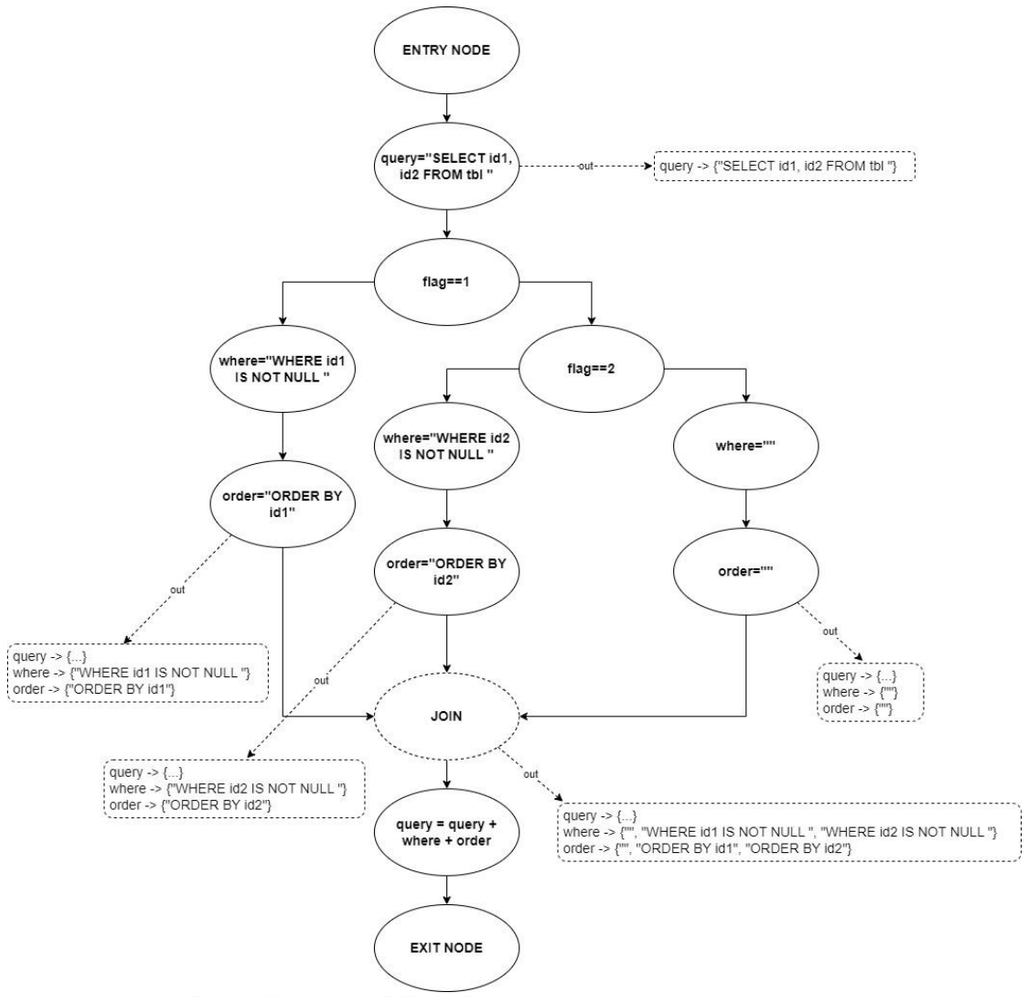

Рис.6. Результат DFA, обчислений стандартним методом

Розглянемо роботу трансферної функції для вузла графу, який передує закінченню програми.

Даний вузол описує операцію об'єднання трьох значень змінних: query, where і order. Вхідна таблиця стану змінних містить наступні значення змінних (таблиця 1).

Таблиця 1. Вхідна таблиця стану змінних

query	where	order
"SELECT id1, id2 FROM tbl "	""	""
	"WHERE id1 IS NOT NULL "	"ORDER BY id1"
	"WHERE id2 IS NOT NULL "	"ORDER BY id2"

Після обчислення декартового добутку множин можливих значень змінних, отримуємо наступні комбінації рядків, які зображено в таблиці 2.

Таблиця 2. Комбінації значень змінних

query	where	order
"SELECT id1, id2 FROM tbl "	""	""
"SELECT id1, id2 FROM tbl "	""	"ORDER BY id1"
"SELECT id1, id2 FROM tbl "	""	"ORDER BY id2"
"SELECT id1, id2 FROM tbl "	"WHERE id1 IS NOT NULL "	""
"SELECT id1, id2 FROM tbl "	"WHERE id1 IS NOT NULL "	"ORDER BY id1"
"SELECT id1, id2 FROM tbl "	"WHERE id1 IS NOT NULL "	"ORDER BY id2"
"SELECT id1, id2 FROM tbl "	"WHERE id2 IS NOT NULL "	""
"SELECT id1, id2 FROM tbl "	"WHERE id2 IS NOT NULL "	"ORDER BY id1"
"SELECT id1, id2 FROM tbl "	"WHERE id2 IS NOT NULL "	"ORDER BY id2"

Шляхом конкатенації всіх комбінацій рядків, які зображено вище, отримуємо наступну множину рядків для змінної query:

1) "SELECT id1, id2 FROM tbl ";
2) "SELECT id1, id2 FROM tbl ORDER BY id1";
3) "SELECT id1, id2 FROM tbl ORDER BY id2";
4) "SELECT id1, id2 FROM tbl WHERE id1 IS NOT NULL ";
5) "SELECT id1, id2 FROM tbl WHERE id1 IS NOT NULL ORDER BY id1";
6) "SELECT id1, id2 FROM tbl WHERE id1 IS NOT NULL ORDER BY id2"
7) "SELECT id1, id2 FROM tbl WHERE id2 IS NOT NULL ";
8) "SELECT id1, id2 FROM tbl WHERE id2 IS NOT NULL ORDER BY id1";
9) "SELECT id1, id2 FROM tbl WHERE id2 IS NOT NULL ORDER BY id2";

Аналізуючи вхідний код програми, можна визначити, що запити під номерами 2, 3, 4, 6, 7, 8 ніколи не будуть створені під час реального виконання інструкцій цієї програми. Інакше кажучи, наявність надлишкових рядків призводить до двох проблем:

1) зменшення точності;
2) комбінаторного вибуху [5] кількості проміжних станів програми, що призводить до значного погіршення швидкодії алгоритму.

На основі аналізу вищеописаних проблем було вирішено розробити модифікований метод статичного аналізу коду для рішення задачі згортки рядкових констант на основі MFP алгоритму, шляхом анотування рядків, які містяться в множині можливих значень змінної, списками компонентів. В даному випадку, компонент – це впорядкована послідовність вузлів графу, що відповідають інструкціям створення рядків, з яких складається значення, що анотується цим компонентом, а список компонентів відповідає операції альтернативи, тобто, якщо список складається з декількох компонентів, то це означає, що такий рядок може бути отриманий декількома шляхами.

Для зберігання значень змінних використовується хеш-таблиця, де ключем хешування є змінна, а значенням є ще одна хеш-таблиця. В цій вкладеній хеш-таблиці ключем хешування є рядок, а значенням – список компонентів. Використання вкладеної хеш-таблиці в якості значень замість структури даних типу «множина», що використовується у стандартному алгоритмі, є відмінною рисою запропонованого методу, яка дозволяє досягти поставленої мети дослідження.

Псевдокод алгоритму нової трансферної функції, що використовується у запропонованому модифікованому методі, показано на рис.7.

```
function transfer (node, dfaIn) {
    stmt = getNodeStmt(node)
    if (getStmtOperation(stmt) === 'assign') {
        to = getAssignTarget(stmt)
        newValues = getAssignValues(stmt)
                    .map(v => isVariable(v) ?
                                dfaIn[v].keyValuePairs() :
                                [v [new Components(node)]])
                    .cartesian()
                    .map(tryGetNewValue)
                    .remove(isNull)
        dfaIn[to] = Map.fromKeyValuePairs(newValues)
    }
    return dfaIn
}
```

Рис.7. Псевдокод модифікованого алгоритму трансферної функції

Опишемо кроки знаходження нового стану змінної.

1. У випадку, якщо сутність - це змінна, то отримується таблиця можливих значень з вхідної таблиці станів змінних, яка трансформується у список пар «рядок – список компонентів». Якщо сутність представлена літералом, стає очевидним, що інструкція створення рядка описана поточним вузлом. Тому, список пар складається з одного елементу, який містить даний літерал і список з одного компонента, що складається з поточного вузла.
2. Обчислюється декартів добуток списків пар, які були отримані на попередньому кроці.
3. Далі розглянемо аналіз, який виконується для кожного елементу добутку, отриманого на попередньому кроці (тобто, для кожної комбінації пар). На початку, обчислюється декартів добуток списків компонентів, що є другими елементами пар. Після цього видаляються ті комбінації, складові (тобто, компоненти) яких є несумісними між собою. Псевдокод алгоритму цього пошуку показаний на рис.8. Компоненти вважаються сумісними, якщо кожен вузол з першого компонента пов'язаний хоча б з одним вузлом другого компонента. Тобто, існує шлях між ними у CFG. Алгоритм перевірки компонентів на сумісність показаний на рис.9. Така перевірка базується на використанні множини попередніх вузлів, яка, як правило, збирається на етапі побудови CFG і міститься у класі вузла в якості окремого поля. У випадку, якщо не знайдено жодної сумісної комбінації компонентів, ця послідовність пар ігнорується. У іншому випадку, кожна сумісна комбінація компонентів перетворюється на єдину компоненту, шляхом поєднання складових даної комбінації.
4. Новий стан змінної, який представлений списком пар «рядок – список компонентів» перетвореним у хеш-таблицю, передається у вихідні властивості вузла (тобто на вхід наступним вузлам).

```
function tryGetNewValue (values) {
    listOfComponents = values.map(v => v[1])
                             .cartesian()
                             .filter(areComponentsCompatible)
                             // об'єднання сумісних компонентів в один
                             .map(list => Components.concat(list))
    return listOfComponents.count > 0 ?
            [values.map(v => v[0]).join(), listOfComponents] :
            null
}
```

Рис.8. Псевдокод алгоритм пошуку сумісних пар

```
function areComponentsCompatible (listOfComponents) {
    for (i = 0; i < listOfComponents.count; i++) {
        for (j = i + 1; j < listOfComponents.count; j++) {
            for (node1 : listOfComponents[i])
                for (node2 : listOfComponents[j])
                    if (!node1.previousNodes.contains(node2) &&
                        !node2.previousNodes.contains(node1))
                        return false
        }
    }
    return true
}
```

Рис.9. Псевдокод алгоритму перевірки компонентів на сумісність

Зазначимо, що у зв'язку із заміною структури даних типу «множина», що використовувалася для зберігання значень змінних у стандартному алгоритмі, хеш-таблицею, змінюється також і join-функція, псевдокод алгоритму якої тепер буде таким, як показано на рис.10. Призначення цієї функції залишається, тобто полягає у поєднанні значень змінної, але в даному випадку, замість об'єднання множин рядків, відбувається злиття хеш-таблиць і конкатенація списків компонентів.

```
function join (in1, in2) {
    return in1.mergeWith(in2,
            (map1, map2) => map1.mergeWith(map2,
                                (list1, list2) => list1.concat(list2)))
}
```

Рис.10. Псевдокод join-функції

Для порівняння запропонованого модифікованого методу із стандартним було створено графік залежності (рис.11) кількості рядків, які можуть бути отримані для змінної query перед закінченням програми, від кількості розгалужень для програми, показаної на рис.3. Цей графік показує, що така залежність для запропонованого модифікованого методу є лінійною, в той час як для стандартної реалізації методу ця залежність є квадратичною. Такий результат вдалося досягти шляхом ігнорування тих комбінацій рядків, які складаються зі значень, що були отримані у різних гілках умовного оператора програми, але які ніколи не з'являться при реальному виконанні програми.

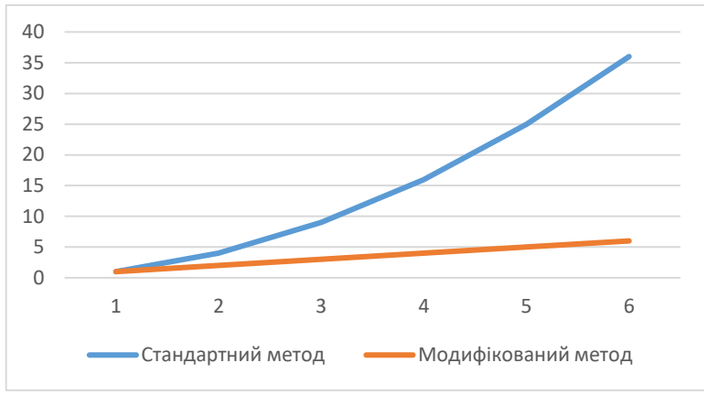

Рис.11. Залежність кількості рядків від кількості розгалужень

Для порівняння запропонованого методу зі стандартним, покажемо відмінності результату аналізу потоку даних (DFA), обчисленого запропонованим методом (рис.12) від стандартного методу (рис.6) для однієї й тієї ж програми (рис.3). Для наочності, вузли графу на рис.12 пронумеровані. На цьому графі списки компонентів зображені квадратними дужками, які містять компоненти, що позначаються або числом (наприклад [1]), де число – це номер вузла, в якому знаходиться рядок, або переліком номерів вузлів, поєднаних амперсандами (наприклад [2&5&6]), якщо рядок даного вузлу був створений шляхом конкатенації інших рядків.

Рис.12 – Результат DFA, обчислений запропонованим методом

Висновки.

В даній статті було запропоновано модифікований метод для рішення задачі згортки рядкових констант, який має меншу складність і вищу точність, ніж існуючі методи. Запропонований метод відрізняється від існуючих використанням вкладеної хеш-таблиці в якості значень основної хеш-таблиці замість структури даних типу «множина» і дозволяє зменшити складність алгоритмів, що використовуються, завдяки ігноруванню тих конкатенацій рядків, які не будуть створені під час реального виконання програми. Ця дія дозволяє зменшити кількість проміжних станів програми у DFA, завдяки чому вдається уникнути комбінаторного вибуху [5] складності, характерного для алгоритмів існуючих методів статичного аналізу для цієї задачі. Крім того, даний модифікований метод також дозволяє збільшити точність звуження теоретичної множини рядків, що можуть бути створені згідно CFG заданої програми, до множини рядків, які фактично будуть створені та використані при виконанні цієї програми.

Напрямками подальших досліджень можуть бути:

1) розширення області застосування запропонованого методу для рішення задачі згортки не тільки рядкових, а також числових та інших видів констант;
2) вдосконалення логіки перевірки значень на сумісність, наприклад, застосуванням аналізу предикатів [6].

Список бібліографічного опису

1. N.Wegman M. Constant propagation with conditional branches [Електронний ресурс] / M. N.Wegman, F. Kcnncth Zadeck // ACM Transactions on Programming Languages and Systems. – 1991. – Режим доступу до ресурсу: https://dl.acm.org/doi/pdf/10.1145/318593.318659.
2. Costantini G. Static Analysis of String Values [Електронний ресурс] / G. Costantini, P. Ferrara, A. Cortesi // International Conference on Formal Engineering Methods. – 2011. – Режим доступу до ресурсу: https://link.springer.com/chapter/10.1007/978-3-642-24559-6_34.
3. Comparison of parser generators [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Comparison_of_parser_generators
4. Control-flow graph [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Control-flow_graph
5. Combinatorial explosion [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Combinatorial_explosion
6. Beyer, D. Combining Model Checking and Data-Flow Analysis [Електронний ресурс] / D. Beyer,, D. Schmidt, S. Gulwani. – 2018. – Режим доступу до ресурсу: https://link.springer.com/chapter/10.1007/978-3-319-10575-8_16.

References

1. N.Wegman M. Constant propagation with conditional branches [Electronic resource] / M. N.Wegman, F. Kenneth Zadeck // ACM Transactions on Programming Languages and Systems. – 1991. – Access mode: https://dl.acm.org/doi/pdf/10.1145/318593.318659.
2. Costantini G. Static Analysis of String Values [Electronic resource] / G. Costantini, P. Ferrara, A. Cortesi // International Conference on Formal Engineering Methods. – 2011. – Access mode: https://link.springer.com/chapter/10.1007/978-3-642-24559-6_34.
3. Comparison of parser generators [Electronic resource] – Access mode: https://en.wikipedia.org/wiki/Comparison_of_parser_generators
4. Control-flow graph [Electronic resource] – Access mode: https://en.wikipedia.org/wiki/Control-flow_graph
5. Combinatorial explosion [Electronic resource] – Access mode: https://en.wikipedia.org/wiki/Combinatorial_explosion
6. Beyer, D. Combining Model Checking and Data-Flow Analysis [Electronic resource] / D. Beyer,, D. Schmidt, S. Gulwani. – 2018. – Access mode: https://link.springer.com/chapter/10.1007/978-3-319-10575-8_16