УДК 004.89
DOI https://doi.org/10.32782/2663-5941/2023.2.1/34

Павлов В.Г.
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

КОНТЕКСТНИЙ ПІДХІД У АНАЛІЗІ СХОЖОСТІ ТЕКСТІВ ПРОГРАМ

У статті розглянути особливості аналізу схожості програмних кодів, як складової захисту інтелектуальної власності, до якої відносяться також й комп'ютерні програми. Ці особливості визначаються граматиками побудови мов програмування, які відносяться до формальних граматик, оскільки ці мови не передбачають виключень та розглядають їх як синтаксичні помилки. Розглянутий механізм компілювання, у якому важливу роль відіграє синтаксичний аналіз. Саме на цьому етапі ланцюжка символів програмного коду трансформується у послідовність лексем, які у свою чергу розпізнаються як токени певних типів. Кількість типів токенів визначається граматикою мови програмування, але серед них можна виділити дві основні групи: стандартні токени та токені користувача. Склад та конструкція перших незмінна в межах мови програмування, тому вони зустрічаються у будь-яких текстах різних програм багато разів. Тому абсолютно не має сенсу порівнювати їх, як звичайний текст, бо завжди будуть збіги. Токени користувача називаються так, бо вони формуються виключно автором програми, і, хоча мають певні обмеження, не повинні збігатися, якщо програмний текст розроблявся автором власноручно та самостійно. Тому збіг серед них є однією з ознак схожості текстів програм. Але часто недоброчесні користувачі намагаються приховати копіювання та вдаються до заміни цих токенів без заміни структури програми. Для виявлення ознак схожості текстів програм в цих випадках треба аналізувати структуру текстів програм, яка складається з певної послідовності та взаємного розташування стандартних токенів, тобто використовувати контекстний підхід.

На підставі проведення експериментів, доведено, що використання для порівняння текстів програм тих же підходів та засобів, що і для звичайних текстів, не дозволяє однозначно знайти в них ознаки схожості. Тому зроблено обґрунтовані висновки щодо перспективності розробки спеціалізованих методик та засобів, які будуть спиратися на синтаксичні особливості мов програмування.

Ключові слова: *інтелектуальна власність, схожість програмного коду, контекстний підхід, граматичний розбір, токен,*

Постановка проблеми. Творчі здібності людини знаходять втілення у її інтелектуальній діяльності, результатом якої є певні здобутки, що відносяться до інтелектуальній власності. Законодавство України чітко визначає, що до об'єктів права інтелектуальної власності відносяться «літературні та художні твори; комп'ютерні програми; компіляції даних (бази даних); …» [1, стаття 420]. Таким чином, на законодавчому рівні закріплюється виключне право авторів «володіти, користуватися і розпоряджатися результатами своєї інтелектуальної, творчої діяльності, які, будучи благом не матеріальним, зберігаються за його творцями і можуть використовуватися іншими особами лише за узгодженням з ними».

Матеріальним втіленням комп'ютерних програм є їх сирцеві тексти, які здебільшого зберігаються у електронному вигляді, що дозволяє досить легко отримати до них доступ недоброчесним користувачам. Тому дуже гостро стоїть питання виявлення та доведення фактів застосування у програмних продуктах ознак використання чужої інтелектуальної власності без дозволу її авторів.

Проблема виявлення схожості у текстових документах відома давно та існує багато підходів щодо її розв'язання, а саме алгоритми пошуку входження одного рядку у інший Бойєра – Мура у різних модифікаціях [2, с. 762-772], алгоритм Кнута – Морріса – Пратта [3, с. 323-350] та інші, але усі вони мають обмежену швидкість та потребують значних обсягів пам'яті для зберігання допоміжних таблиць. Також у багатьох алгоритмах пошуку схожості використовується відомий метод «шинглів» (shingles), запропонований А. Бродером у 1997 році [4, с. 28], який дозволяє підвищити швидкість пошуку за рахунок використання паралельної обробки декількох блоків тексту. Але усі алгоритми розглядають об'єкт пошуку та порівняння як звичайний текст, тобто мають універсальне використання, яке не враховує та не використовує специфіки мов програмування, їх граматику та синтаксис.

Метою статті є викладення граматичних особливостей мов програмування та розгляд їх використання у контекстному підході до аналізу схожості текстів програм. Для досягнення означеної мети вирішуються наступні завдання:

– визначення граматичних особливостей побудови мов програмування;

– використання цих особливостей у аналізі схожості текстів програм.

Виклад основного матеріалу дослідження. Побудова будь-якої мови ґрунтується на множині певних правил, які складають граматику цієї мови. Мови спілкування людей формувалися стихійно, оскільки у них виникала гостра потреба, а граматики формувалися постфактум і часто описували вже існуючі природні мови. Внаслідок цього часто виникали протиріччя між фактичним застосуванням певних мовних фразеологізмів та правилами розробленої граматики, через що у багатьох мовах спілкування з'являлися виключення з правил.

Мови програмування базуються на граматиках, які не мають виключень, та завжди формують сталі конструкції, оскільки на підставі цих правил будуються розпізнавачі у складі компіляторів. Будь яка невідповідність правилам граматики трактується як синтаксична помилка та обробляється компіляторами на етапі розпізнавання тексту. Тому мови програмування, як й інші мови, які створювалися штучно, відносяться до формальних мов, а їх граматики до формальних граматик. Лінгвістична теорія таких мов, яка була розроблена Ноамом Хомскі [5, с. 18-25], описувала ієрархічну структуру, що складалася з чотирьох класів породжуючих граматик. В залежності від класу, до якого належала граматика, вона мала описуватися тільки правилами визначеного типу, що відповідало побудові розпізнавача для цієї мови у складі компіляторів.

Таким чином, будь яка мова програмування на відміну від природних мов має сталу множину граматичних конструкції, яких набагато менше ніж у природних мовах, бо інакше процес компілювання не міг би бути обмежений у часі. Розпізнавач під час граматичного розбору або парсінгу робить певну множину кроків в залежності від кількості гілок у синтаксичному дереві. Як тільки йому вдається побудувати ланцюжку правил граматики, за допомогою яких можна побудувати (породити) певну конструкцію, вона вважається розпізнаною. Усі нерозпізнані конструкції вважаються помилковими і не підлягають компіляції. Оскільки тексти програм, які аналізуються на предмет схожості, не повинні містить синтаксичні помилки, то усі конструкції повинні відповідати правилам граматики певної мови програмування, а їх складові належать абетці цієї мови.

Розглянемо послідовність обробки будь-якої послідовності символів під час компіляції. Вона зазвичай складається з наступних етапів [6, с. 3]:

– лексичний аналіз, під час якого первісна послідовність розділяється на окремі складові – лексеми, в свою чергу лексеми розподіляються по типам и формують токени – елементарні стали конструкції мови;

– синтаксичний аналіз, при якому аналізуються сполучення токенів, їх послідовність та поєднання, які повинні відповідати правилам граматики даної мови програмування;

– генерація коду, коли кожний токен або їх певне сполучення замінюється фрагментом машинного коду.

Розглянемо більш детально процес формування з лексем токенів, який нагадує розподіл лексем по певним типам. У кожній мові програмування існують певні ***стандартні токени***, наприклад:

– ключеві слова (назви команд, операторів, функції тощо);

– знаки математичних операції;

– знаки логічних операції;

– дужки;

– знаки розділення та пунктуації.

Усі стандартні токени заздалегідь відомі компілятору, бо вони містяться у ньому у спеціальних таблицях. Кожна лексема спочатку порівнюється зі стандартними токенами зі таблиць. Якщо лексема співпала з якімось стандартним токеном зі таблиці, то вона ідентифікується у програмі як токен певного типу.

Усі ті токени, які не співпали зі стандартними, але задовольняють граматиці мови, відносяться до токенів, що створює програміст, наприклад:

– ідентифікатори (назви змінних);

– числові та символьні літерали (константи).

– мітки.

Звичайно, вони можуть бути різними у різних програмах, написаних на одній і тієї ж мові, бо це цілком залежать від фантазії програміста, тому не можуть бути присутніми у таблицях стандартних токенів, а створюють свою таблицю і додаються до неї вже безпосередньо під час лексичного аналізу. Будемо ці токені у подальшому називати ***токенами користувача*** або користувацькими.

Саме наявність у складі текстів програм токенів цих двох типів ї визначає їх граматичну особливість у порівнянні зі звичайними текстами, а саме:

– наявність у текстах, що порівнюються, повторень або збігів стандартних токенів не є безперечною ознакою наявності їх схожості;

– наявність збігів у токенах користувача, навпаки, є безперечною ознакою такої схожості, бо вірогідність, що довільні назви будуть однаковими майже нульова, якщо авторами текстів програм були різні люди.

Ці дві важливі особливості граматик текстів програм можна покласти у основу стратегії аналізу схожості текстів програм, а саме:

1. Стандартні токени позначають певний каркас текстів програм, до якого «прив'язано» розташування токенів користувача.

2. Відмінності виключно лише у користувацьких токенах при однакових «каркасах» програм можна вважати ознакою схожості.

Для ілюстрації цього факту можна привести практику заміни назв змінних у текстах програм при збереженні послідовності стандартних токенів.

Наприклад у фрагменті тексту програми на мові С:

```
for (int i = j; j >= 0; i--)
{
if (input[i] == "=")
{
exitCode.Append(" " + input[i – 1]);
break;
}
}
```

внесені зміни шляхом взаємної заміни назв ідентифікаторів ***i*** та ***j***:

```
for (int j = i; i >= 0; j--)
{
if (input[j] == "=")
{
exitCode.Append(" " + input[j – 1]);
break;
}
}
```

Зрозуміло, що з позиції програмування ці два фрагменти мають ознаки 100% схожості, але якщо до цих фрагментів коду застосувати той же підхід, що й до фрагментів звичайного тексту, то результат буде зовсім іншим. Для проведення експерименту були вибрані декілька безкоштовних програмних продуктів для порівняння двох текстів у он-лайн режимі. Тому у тестуванні не використовувалися такі програмні засоби, як UNICHECK (https://unicheck.com), Edu-Birde (https://edubirdie.com), **ABBYY Comparator** (https://www.abbyy.com), **WinMerge** (https://winmerge.org), **Araxis Merge** (araxis.com/merge/), Etxt Antiplagiat (https://www.etxt.ru), Text.ru (https://text.ru/), **Shingles Expert** (http://makebusiness.ru), **TextDiff** (http://www.angusj.com), **Kaleidoscope** (kaleidoscopeapp.com), **Beyond Compare** (https://www.scootersoftware.com/download.php), **DeltaWalker** (deltawalker.com), **Gedit** (https://wiki.gnome.org/Apps/Gedit), **Vim** (https://www.vim.org), бо вони або потребували інсталяції, реєстрації або сплати за використання.

Вказані фрагменти програмного коду порівнювалися за допомогою таких 9 програм: **DiffChecker** (https://www.diffchecker.com/text-compare/), **Charactercalculator** (https://charactercalculator.com/ru/text-compare/), **Leaubk** (https://leaubk.com/2_text), **Draftable** (https://draftable.com/compare), **Pr-cy** (https://pr-cy.ru/difference/), **Copyleaks** (https://app.copyleaks.com/ru/text-compare), **Anytexteditor** (https://anytexteditor.com/ru/text-compare), **Smodin** (https://smodin.io/uk/), **Siteanalyzer** (https://site-analyzer.pro/uk). Усі вони успішно знаходили та помічали розбіжності у фрагментах програм, але лише чотири останні з них робили спробу якимось чином кількісно оцінити ці розбіжності. На жаль, розробники більшості подібних програм не надають інформацію яким чином проводяться ці кількісні оцінки, лише у двох з них (**Copyleaks** та **Anytexteditor**) можна зрозуміти, що у оцінюванні ступені схожості враховується відносна кількість виявлених відмінностей при посимвольному порівнянні, а у програмі **Siteanalyzer** надається інформація, що при порівнянні використовуються «шингли» розміром у 4 слова. За підсумками тестування були отримані наступні результати:

- **Draftable** – 72% схожості;
- **Anytexteditor** – 87.96% схожості (відрізняються 13 символів зі 108);
- **Smodin** – 9% плагіату;
- **Siteanalyzer –** тексти схожі на 0%, тому що жодний «шингл» не співпав.

Якщо у перші дві програми дають оцінки схожості хоча б якось наближені до дійсності, то дві останні дуже важко пояснити логічно, бо нагадаємо, що тестові тексти з точки зору відповідності програмних кодів ***не мають відмінностей***, тобто схожість складає **100%**.

Таким чином, шляхом тестування програм, що визначають схожість текстів, експериментально доведено, що вони ***надають хибну інформацію, а тому не можуть бути застосовані для порівняння програмних кодів***. З цього випливає, що застосування до аналізу схожості програмних кодів тих же підходів та алгоритмів, що й до звичайних текстових документів, ***неправомірно***, а, відповідно, повинні застосовуватися спеціальні

методи, основані на граматичних особливостях мов програмування.

Другий експеримент, який було проведено з іншими фрагментами програмного коду на мові **C**, був спрямований продемонструвати, що програми порівняння текстів не враховують синтаксичний контекст фрагментів, які аналізуються.

Нехай треба порахувати суму $S=\sum_{i=1}^{10}\frac{1}{i^2}$. Зрозуміло, що для цього у програмі повинен бути організований цикл, у якому змінна ***i*** буде послідовно приймати значення від 1 до 10. Але існує декілька способів організації таких циклів, наприклад покроковий цикл, цикл з передумовою та цикл з післяумовою. Для двох останніх випадків фрагменти програмного коду будуть мати вигляди:

- з передумовою:

```
i=1;
s=0;
while(i<=10)
{
 s=s+1/(i*i);
 i=i+1;
}
```

- з післямовою:

```
 i=1;
 s=0;
do
 {
 s=s+1/(i*i);
 i=i+1;
 }
while (i<=10);
```

За підсумками порівняння фрагментів текстів цих програм отримано:

- **Draftable** – 85,7% схожості;
- **Anytexteditor** – 46.75 схожості (відрізняються 36 символів зі 77);
- **Smodin** – 37% плагіату;
- **Siteanalyzer –** тексти схожі на 0%, але знову причина у тому, що жодний «шингл» не співпав.

У наведеному прикладі у фрагментах коду використовуються ***одні й ти ж самі назви змінних та арифметичні оператори***, але ***різні*** конструкції операторів циклу***,*** які мають ***різну синтаксичну побудову***. Тобто при компіляції цих фрагментів розпізнавачі будуть будувати ***різні*** синтаксичні дерева, тож є безумовні підстави вважати ці фрагменти кодів ***різними***. Але програмні засоби, які виконують аналіз схожості, не розрізняють відмінності у стандартних і користувацьких токенах, тому надають спотворений результат порівняння, якій не відповідає дійсності, а тому програми порівняння текстів не підходять для програмних кодів.

Висновки. Отримані результати аналізу схожості текстів програм за допомогою засобів, які використовують той же підхід, що й до звичайних текстів, дають підстави зробити наступні висновки:

1. Стандартній підхід до визначення схожості звичайних текстів не може бути використаний для порівняння текстів програм, оскільки не враховує синтаксичні особливості побудови мов програмування.

2. При аналізі програмних кодів треба враховувати, що тексти програм складаються з токенів двох типів, тому збіги у стандартних токенах не є ознакою схожості, у той час, як збіги у користувацьких токенах є певною її ознакою.

3. Важливою ознакою для прийняття рішення щодо наявності або відсутності схожості тексті програм є послідовність та взаємне розташування у коді стандартних токенів з однаковим розміщенням відносно них токенів користувача, тобто контекстний підхід.

Список літератури:

1. Цивільний кодекс України: Закон України від 16.01.2003 № 435-IV. *Відомості Верховної Ради України, 2003, №№ 40-44, ст.356.* Дата оновлення: 28.03.2023. URL: https://zakon.rada.gov.ua/laws/show/435-15#Text (дата звернення 29.03.2023).

2. Boyer R. S., Moore J. S. A fast string searching algorithm. *Communication of the ACM*. 1977. V. 20. No 10. P. 762–772.

3. Knuth D. E., Morris J. H., Jr., Pratt V. R. Fast pattern matching in strings. *SIAM Journal on Computing*. 1977. V. 6. No 2, P. 323–350. DOI: https://doi.org/10.1137/0206024.

4. Broder A. Z. On the resemblance and containment of documents. *Proceedings. Compression and Complexity of SEQUENCES 1997 (Salerno, Italy13-13 June 1997)*. IEEE Computer Society, 1998. P. 21–29. DOI: https://doi.org/ 10.1109/SEQUEN.1997.666900.

5. Chomsky N. Syntactic structures. Copyright 1957, Second Edition, Berlin: Mouton de Gruyter, 2002. 117 p.

6. Pavlov V. Nullifying rules influence on speed in context free grammar LL(1). Journal of Theoretical and Applied Computer Science. Polish Academy of Sciences, Gdansk Branch, Computer Science Commission. 2016. V2. P. 3–15.

Pavlov V.G. CONTEXTUAL APPROACH IN ANALYZING THE SIMILARITY OF PROGRAM CODES

In the article to consider the features of the analysis of the similarity of program codes, as a component of the protection of intellectual property, which also includes computer programs. These features are determined by the grammars of constructing programming languages that relate to formal grammars, since these languages do not involve exceptions and treat them as syntactic errors. The compilation mechanism in which syntactic analysis plays an important role is considered. It is at this stage that the chain of characters of the program code is transformed into a sequence of lexemes, which in turn are recognized as tokens of certain types. The number of token types is determined by the grammar of the programming language, but among them are two main groups: standard tokens and user tokens. The composition and construction of the former is unchanged within the programming language, so they are found in any texts of different programs many times. Therefore, it absolutely does not make sense to compare them like plain text, because there will always be coincidences. User tokens are called so because they are formed solely by the author of the program, and, although they have certain limitations, should not coincide, if the program text was developed by the author personally and independently. Therefore, the coincidence among them is one of the signs of the similarity of the texts of programs. But often dishonest users try to hide the copying and resort to replacing these tokens without replacing the program structure. To identify signs of similarity of program texts in these cases, it is necessary to analyze the structure of the texts of programs, which consists of a certain sequence and relative arrangement of standard tokens, that is, to use a contextual approach.

Based on the experiments, it is proved that the use of the same approaches and tools for comparing texts of programs as for ordinary texts, does not allow to find signs of similarity in them unambiguously. Therefore, reasonable conclusions have been made regarding the prospects of developing specialized methods and means, which will rely on the syntactic features of programming languages.

Key words: *intellectual property, similarity of program codes, contextual approach, grammatical parse, token.*