Науковий вісник НЛТУ України
Scientific Bulletin of UNFU
https://nv.nltu.edu.ua
https://doi.org/10.36930/40310515
Article received 16.11.2021 p.
Article accepted 25.11.2021 p.
UDC 004.415.532.3

ISSN 1994-7836 (print)
ISSN 2519-2477 (online)

@ Correspondence author
A. V. Franko
andrii.v.franko@lpnu.ua

Р. П. Базилевич, А. В. Франко
Національний університет "Львівська політехніка", м. Львів, Україна

ІЄРАРХІЧНА МОДЕЛЬ СИСТЕМ АВТОМАТИЗОВАНОГО ГЕНЕРУВАННЯ МОДУЛЬНИХ ТЕСТІВ

Описано особливості проблеми тестування програмного забезпечення (ПЗ) за допомогою автоматизованих систем генерування модульних тестів. Проаналізовано методи автоматизованого модульного тестування, що використовуються для тестування ПЗ. Виконано класифікацію методів генерування модульних тестів на підставі вхідних даних і засобів для генерування тестів. Показано, що компільований байт-код та граф контролю потоку є основними видами вхідних даних, а символьне виконання є основним методом для генерування модульних тестів. Систематизовано новітні методи автоматизованого модульного тестування: символьне виконання з використанням штучних нейронних мереж, додаткової логіки та оптимізаційних алгоритмів. Проаналізовано можливості застосування мета- та гіперевристик системами автоматизованого генерування модульних тестів. Побудовано їх ієрархічну модель: до четвертого рівня віднесено пошукові алгоритми для аналізу умов у коді; до третього – SMT-бібліотеки, які містять множину алгоритмів першого рівня та стратегії їх використання; до другого – поєднання результатів роботи SMT-бібліотеки з результатами роботи додаткової логіки; до першого – алгоритм управління, що керують процесом генерування тестів. Описано можливості виконання паралельних обчислень на всіх рівнях ієрархії. Продемонстровано наявність вузьких місць у реалізаціях систем генерування модульних тестів. Запропоновано розподіл завдання генерування модульних тестів на підставі рівнів ієрархії моделі, що дає змогу обійти вузькі місця поточних систем та покращити масштабованість. Розроблено UML-діаграму класів на запропонованій моделі. Запропоновано одночасне використання метаевристик на всіх ієрархічних рівнях моделі для підвищення якості згенерованих тестів, що покращить універсальність і модульність системи. Обґрунтовано потребу подальшого розроблення нових методів для підвищення ефективності алгоритмів генерування тестів та якості тестування.

Ключові слова: модульне тестування; символьне виконання; ефективність обчислень; якість програмного забезпечення; евристичні алгоритми.

Вступ / Introduction

Основна причина збоїв у роботі ПЗ, які спричинені помилками імплементації – це недостатність тестування, обмеження процесів тестування у часі та ресурсах, також людський чинник. Код і дизайн ПЗ не піддаються повному тестуванню з таких причин [11]: 1) неможливо протестувати всі можливі види вхідних даних; 2) неможливо протестувати всі можливі комбінації вхідних даних; 3) неможливо протестувати всі шляхи виконання програми; 4) неможливо виявити всі помилки дизайну інтерфейсу та аналізу вимог; 5) неможливо відтворити всі реальні умови використання програмного продукту.

Для забезпечення якості ПЗ використовують методи автоматизованого генерування модульних тестів. У розробках автоматизованих методів генерування модульних тестів досягнуто значного прогресу, зокрема створено методи та засоби символьного виконання [1, 7, 10, 12, 16, 17, 20, 21, 22], які генерують тести для коду, написаного на C, C++, Java та інших мовах програмування [2, 4, 5, 8, 13, 14, 15, 18]. Наявні засоби використовують оптимізаційні алгоритми [2, 6, 9], штучні нейронні мереж та інші засоби [10, 18]. Обчислювальну ефективність покращено завдяки паралелізації алгоритмів [3, 19, 20]. Наявні методи автоматизованого генерування модульних тестів є спеціалізованими, що відкриває простір для застосування метаевристик, що потребують значного процесорного часу. Актуальним є створення нових моделей і методів для застосування мета- та гіпереристик та підвищення швидкості обчислень.

Об'єкт дослідження – автоматизоване генерування модульних тестів на підставі початкового коду ПЗ.

Предмет дослідження – методи автоматизованого генерування модульних тестів на підставі аналізу початкового коду ПЗ, що базуються на символьному виконанні та його варіаціях.

Мета роботи – розробити ієрархічну модель систем автоматизованого генерування модульних тестів на підставі початкового коду ПЗ, що дасть змогу підвищити швидкість та якість його тестування завдяки застосуванню запропонованої моделі та архітектури на її підставі.

Інформація про авторів:
Базилевич Роман Петрович, д-р техн. наук, професор, кафедра програмного забезпечення. **Email:** rbaz@polynet.lviv.ua; https://orcid.org/0000-0002-7949-1353
Франко Андрій Вадимович, аспірант, кафедра програмного забезпечення. **Email:** andrii.v.franko@lpnu.ua; https://orcid.org/0000-0002-8359-7353

Цитування за ДСТУ: Базилевич Р. П., Франко А. В. Ієрархічна модель систем автоматизованого генерування модульних тестів. Науковий вісник НЛТУ України. 2021, т. 31, № 5. С. 96–101.
Citation APA: Bazylevych, R. P., & Franko, A. V. (2021). Hierarchical model of automated test generation system. *Scientific Bulletin of UNFU, 31*(5), 96–101. https://doi.org/10.36930/40310515

Для досягнення зазначеної мети визначено такі основні *завдання дослідження*:

1. Здійснити аналіз та класифікацію методів автоматизованого генерування модульних тестів;
2. Узагальнити архітектуру системи та моделей, які використовуються для автоматизованого генерування модульних тестів;
3. Створити універсальну модель системи автоматизованого генерування модульних тестів;
4. Проаналізувати варіанти застосування запропонованої моделі для покращення ефективності обчислень та показників покриття коду генерованими тестами.

Наукова новизна отриманих результатів дослідження – вперше запропоновано універсальну ієрархічну модель для систем автоматизованого генерування модульних тестів і архітектуру ПЗ на її підставі, що дасть змогу значно підвищити швидкість та якість його тестування завдяки застосуванню запропонованої моделі та архітектури на її підставі.

Практична значущість результатів дослідження – систематизація систем і методів автоматизованого генерування модульних тестів дає змогу створити нові моделі для паралельних обчислень, що необхідно для пришвидшення процесу розроблення ПЗ та підвищення його якості; запропоновані варіанти використання сформованої моделі підвищать покриття коду згенерованими тестами та обчислювальну ефективність систем автоматизованого модульного тестування.

Аналіз останніх досліджень та публікацій. Ідеї автоматизованого генерування модульних тестів розвивалися в напрямках [1, 13, 14, 16, 22]:

1) Символьне виконання. Основні проблеми, які намагалися вирішити:
   - дуже швидкий ріст кількості шляхів для аналізу;
   - неповна відповідність модельованого шляху до реального;
   - велика складність обчислення для розрахунку вхідних даних;
   - відсутність алгоритмів для аналізу деяких символьних обмежень.
2) Тестування на підставі моделей ПЗ. Розроблялися методи створення та аналізу моделі на підставі можливих станів. З великого простору можливих вхідних даних та конфігурацій вибирався їх обмежений набір, що забезпечує значне покриття тестами заданих функцій ПЗ.
3) Адаптивне випадкове тестування. Пропонуються методи тестування з використанням додаткових алгоритмів, що адаптують випадковий вибір до потреб конкретної функціональності.
4) Автоматизоване тестування з використанням оптимізаційних (пошукових) алгоритмів. Розробляються еволюційні алгоритми генерування тестів зі застосуванням фітнес-функції, яка забезпечує вибір вхідних даних, що задовольняють критерії покриття коду.
5) Генерування вхідних даних з використанням алгоритмів локальної оптимізації.

Засоби символьного виконання використовують спеціальне ПЗ для аналізу логіки програмного коду, а також механізми, які є подібними до компіляторів, або модифіковані компілятори для перетворення коду у граф потоку керування або іншу зручну для аналізу структуру. Для аналізу комбінацій умовних операторів (символьних обмежень) використовуються методи для вирішення задачі SAT (виконуваності булових формул) та узагальнення задачі SMT (виконуваності формул) [2, 8, 12, 21]. Задачі SAT вирішуються за допомогою пошукового алгоритму DPLL (Davis – Putnam – Logemann – Loveland) або CDCL (conflict-driven clause learning) [2, 8, 12, 21]. Задачі SMT зводяться до задач SAT. Прикладами бібліотек для вирішення SAT та SMT є Z3-solver [2], Reluplex [12], Yikes [8], SMT-RAT, ABsolver, Barcelogic, Beaver, Boolector та інші. Основними можливостями такого ПЗ є знаходження значень для вільних змінних у формулі. SMT-вирішувачі піддаються паралелізації [21] та забезпечують значне пришвидшення завдяки декомпозиції задачі.

Для вдосконалення символьного аналізу пропонується використовувати нейронні мережі [18], які дають змогу краще справлятися з експоненційним зростаням кількості шляхів та недоліками SMT-вирішувачів, обмеженнями аналізу коду. За допомогою нейронної мережі будується апроксимована репрезентація логіки програми. Така апроксимація виконується для кожного фрагмента програми окремо [10, 18], що дає змогу враховувати логічні зв'язки між сутностями, які були виявлені при апроксимації ділянок коду. Для об'єднання результатів роботи нейромережі та методів символьного виконання створюються та вирішуються комбіновані обмеження. Пошук рішення для таких обмежень передбачає задоволення як обмежень ідентифікованих методами символьного виконання, так і нейромережевих обмежень.

Символьне виконання оптимізують завдяки інструментизації початкового коду та використання компіляції замість інтерпретації [4, 5, 7, 15, 17]. Символьне виконання також поєднують з оптимізаційними методами. Такими прикладами є системи Austin [13] та Ocelot [16]. Використовуються такі алгоритми: моделювання відпалу, сходження вгору, забороненого пошуку, генетичні, імітації колективної поведінки живих організмів та інші [6]. Генератори тестів, що використовують символьне виконання для аналізу коду, можуть використовувати генетичні алгоритми для керування його процесом [9], такі методи називають комбінованими. Добрим генератором тестів, що використовує символьне виконання, кероване генетичним алгоритмом для мови Java, є EvoSuit [6]. Цей засіб оптимізує критерії покриття для всього набору тестів, а не тільки для одного тесту [9]. Для генерування вхідних даних для тестів в EvoSuit використовуються локальні оптимізаційні алгоритми та динамічне символьне виконання [6].

Результати дослідження та їх обговорення / Research results and their discussion

Класифікація методів генерування модульних тестів. Для узагальнення автоматизованих систем генерування модульних тестів необхідно їх класифікувати на підставі вхідних даних. Для генерування тестів використовуються вимоги та документація (UML-діаграми), моделі ПЗ, початковий код та інші дані, які можуть застосовуватися сумісно. На рис. 1 згруповано методи автоматизованого генерування модульних тестів на підставі типу вхідних даних. Комбіновані методи можуть використовувати декілька видів вхідних даних сумісно. Використання декількох типів вхідних даних одночасно є найбільш перспективним. Серед розглянутих засобів [1, 3, 4, 5, 6, 9, 13, 14, 16, 18, 15, 22] найпопулярнішими вхідними даними є байт-код і граф контролю потоку, згенерований на підставі коду ПЗ.

Існує декілька варіантів застосування автоматизованого генерування тестів: фазинг тестування, знімок по-

точного стану коду та перевірка коду на відповідності моделі чи протоколу. Використовуються також класифікації за типом проблеми: тестування для підвищення надійності, тестування протоколу чи алгоритму, тестування функціональності та інші. Розроблено різні стратегії вибору гілок для аналізу з використанням декількох критеріїв: покриттям гілок, покриттям операторів, покриттям лінійних послідовностей операцій та переходів та інші.

Додатково класифікувати методи аналізу коду при символьному тестуванні можна за: критерієм достатності тестування; методами статичного та динамічного аналізу; алгоритмами аналізу вхідних даних; типом вхідних даних; метою застосування (рис. 2).

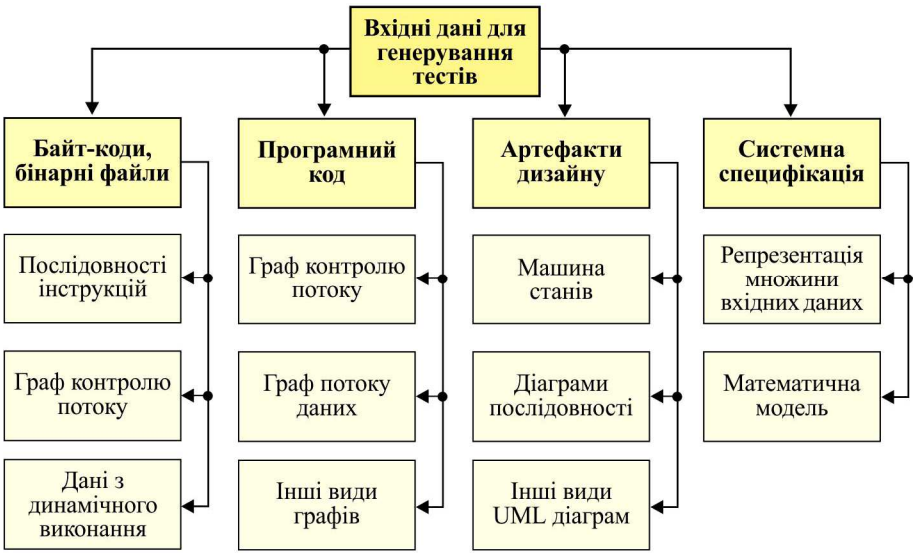

Рис. 1. Класифікація методів автоматизованого генерування модульних тестів на підставі типу вхідних даних / Classification of methods for automated unit test generation based on required input data

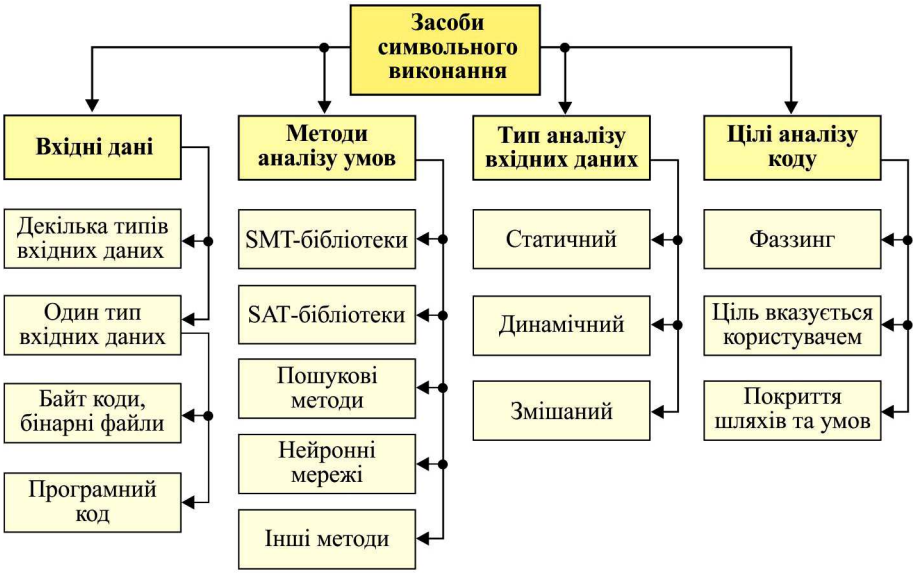

Рис. 2. Класифікація методів аналізу коду при символьному тестуванні / Classification of code analysis methods used for the symbolic execution technique

Ієрархічна модель структури системи для автоматизованого модульного тестування. На підставі узагальнення попередніх часткових випадків створено узагальнену модель системи для автоматизованого модульного тестування (рис. 3), що містить чотири рівні. На четвертому рівні розташовані оптимізаційні алгоритми, що використовуються для аналізу символьних обмежень. При подібному аналізі доступний паралелізм, оскільки декілька обмежень можуть аналізуватися одночасно, та одне обмеження може аналізувати декілька алгоритмів. На третьому рівні розташовані бібліотеки, утворені множинами алгоритмів та стратегій для вирішення символьних обмежень. Декілька бібліотек можуть використовуватися одночасно для аналізу обмежень та підвищення якості отриманих результатів. На другому рівні системи поєднуються результати аналізу символьних обмежень і додаткових правил системи.

Додаткові правила та логіка представлені нейромережами, генетичними алгоритмами і додатковими логічними умовами для перевірки. На першому рівні знаходиться алгоритм керування, що відповідає за логіку створення тестів, аналізує близькість до заданих критеріїв тестування та задає вхідні дані для інших рівнів. Сучасні системи для генерування тестів є складними, ієрархічним та використовують різноманітні оптимізаційні алгоритми, в т.ч. мета- та гіперевристики, засоби штучного інтелекту, що зумовлюють додаткову складність, проте покращують якість тестування. У них існу-

ють відношення типу багато до багатьох між SMT-вирішувачами та цими алгоритмами і ПЗ для генерування модульних тестів. Ієрархічна структура системи дає змогу розпаралелювати процес тестування на кожному її рівні.

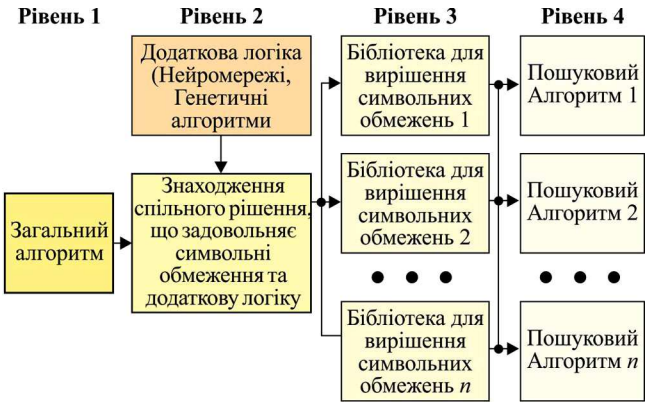

Рис. 3. Узагальнена ієрархічна модель структури систем для автоматизованого модульного тестування / The generalized hierarchical model of the automated unit test generator

Застосування ієрархічної моделі для декомпозиції задачі автоматизованого модульного тестування та паралельних обчислень. Розроблена модель призначена для підвищення ефективності та якості тестування. Такі засоби генерування модульних тестів потребують значних часових затрат, оскільки завдання генерування тестів зводиться до SMT та SAT задач, що вимагають неполіноміального часу обчислень [2], та наявність великого об'єму вхідних даних. Використовуючи узагальнену структуру систем автоматизованого модульного тестування можна створити нові варіанти архітектури ПЗ для автоматизованого генерування модульних тестів.

Нагажаємо, що паралельні обчислення – форма обчислень, в яких кілька дій проводяться одночасно. Ґрунтуються на тому, що великі задачі можна розділити на декілька менших, кожну з яких можна розв'язати незалежно від інших. Є кілька різних рівнів паралельних обчислень: бітовий, інструкцій, даних та паралелізм задач. Паралельні обчислення застосовують вже протягом багатьох років, в основному в високопродуктивних обчисленнях, але зацікавлення ним зросло тільки недавно через фізичні обмеження зростання частоти. Оскільки споживана потужність (і відповідно виділення тепла) комп'ютерами стало проблемою в останні роки, паралельне програмування стає домінуючою парадигмою в комп'ютерній архітектурі, основному в формі багатоядерних процесорів.

На рис. 4 подано діаграму класів, що розроблена на підставі ієрархічної моделі системи автоматизованого генерування тестів. На відміну від наявних систем з розбиттям на обчислювальні вузли, що мають повний функціонал системи, пропонується додатково розподілити обчислювальні вузли за ієрархічними рівнями в системі. З'явиться додаткова можливість балансувати навантаження не тільки розподіляючи вхідні дані між вузлами, а й виділяючи додаткові обчислювальні вузли на кожному ієрархічному рівні моделі. Такий розподіл сприятиме більшою мірою використати множинне відношення багато до багатьох, що склалося в області автоматизованого модульного тестування. Також очікується збільшення кількості комунікаційних повідомлень між обчислювальним вузлами, що є недоліком запропонованої моделі. Проте виграш від можливості збільшувати обчислювальні потужності для подолання вузьких місць буде кращим, ніж додаткові витрати на комунікацію між ієрархічними рівнями.

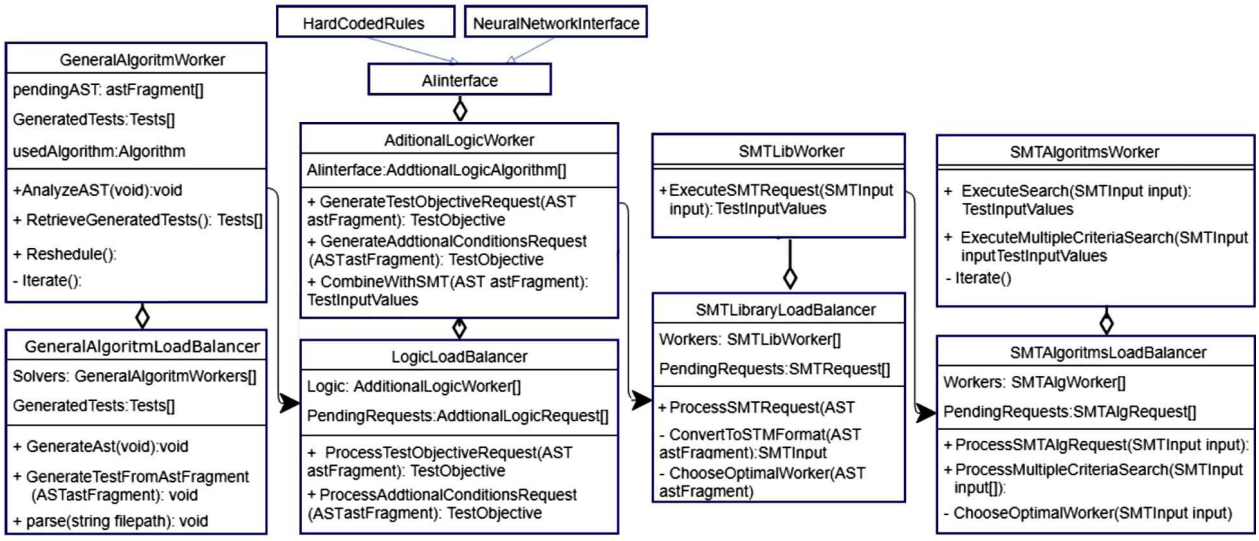

Рис. 4. Структурна діаграма класів на ієрархічній моделі / Structural UML class diagram based on the hierarchical model

Паралельне використання метаевристик на всіх ієрархічних рівнях моделі очікувано підвищить покриття коду тестами та дасть змогу покривати ділянки, які раніше не покривалися тестами через недоліки та дефекти окремих алгоритмів. Завдяки використанню декількох алгоритмів паралельно можна частково подолати відомі недоліки символьного виконання. Метаевристичний підхід, систематично застосований для подолання вузьких місць, підвищить кількість тестів та покриття коду.

На підставі запропонованої ієрархічної моделі системи автоматизованого модульного тестування можливо розробити нові архітектури для паралельних обчислень. Критерії для оцінки повинні включати показники ефективності обчислень, використання електроенергії, покриття коду згенерованими тестами.

Обговорення результатів дослідження. Наявні системи [3, 6, 9, 14, 19, 20] використовують декомпозицію задачі на підставі вхідних даних, що призводить до формування нерівномірного розподілу задач між обчис-

лювальними процесами. У системах [3, 6, 9, 19, 20] містяться такі структурні елементи:

1) керівний вузол – відповідає першому рівню ієрархічної моделі (може бути розділений на функції контролю процесу обчислень та надання вхідних даних);
2) обчислювальні вузли – відповідають 2-4 рівням ієрархічної моделі.

Особливості обчислювальних сутностей у системах [3, 6, 9, 14, 19, 20] не створюють можливості повноцінно збалансувати систему при вхідних даних, що утворюють незбалансоване абстрактне синтаксичне дерево. На рис. 5 схематично зображено рівномірно розподілені вхідні дані порівняно із вхідними даними з дисбалансом. Реальні вхідні дані не є збалансованими деревами, під час їх аналізу часто виникають "вузькі місця". Для виникнення вузького місця необхідні такі умови:

1) абстрактне синтаксичне дерево є незбалансованим;
2) велика кількість вузлів у абстрактному синтаксичному дереві пов'язана з декількома вузлами, що є складним для аналізу умовними операторами.

Нагадаємо, що абстрактне синтаксичне дерево (англ. *Abstract Syntax Tree*) – це скінченна множина, позначене й орієнтоване дерево, в якому внутрішні вершини співставлені з відповідними операторами мови програмування, а листя з відповідними операндами. Синтаксичні дерева використовують в парсерах для проміжного подання програми між деревом розбору (конкретним синтаксичним деревом) і структурою даних, яку потім використовують як внутрішнє подання компілятора або інтерпретатора комп'ютерної програми для оптимізації та генерації коду. Можливі варіанти подібних структур описують абстрактним синтаксисом.

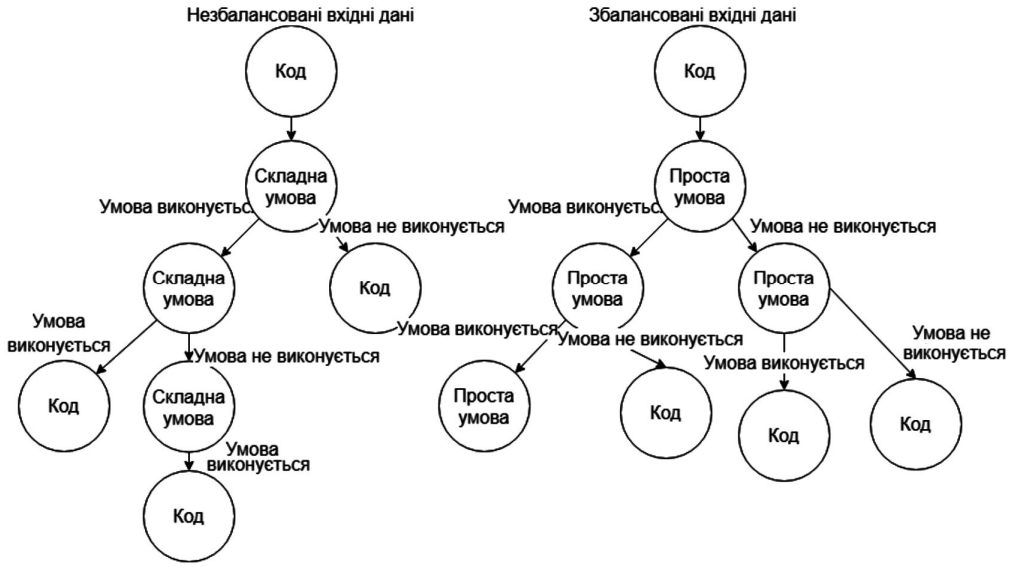

Рис. 5. Збалансовані (права сторона рисунка) та незбалансовані (ліва сторона рисунка) вхідні дані / Balanced (on the right) and unbalanced (on the left) input data

Паралельні обчислення використовуються у SMT-бібліотеках [2] на різних рівнях, як для окремих алгоритмів, так і для декількох SMT-бібліотек одночасно. Сучасні засоби не застосовують всіх можливостей SMT-бібліотек [3, 6, 9, 14, 19, 20], запропонована модель дасть змогу ширше використовувати багатокритеріальний пошук та можливості паралельних обчислень SMT- бібліотек.

Висновок / Conclusions

Систематизовано та узагальнено процес автоматизованого генерування модульних тестів, у т.ч. символьне виконання зокрема з використанням штучних нейронних мереж та додаткової логіки, з інструментизацією коду та з використанням оптимізаційних (пошукових) алгоритмів.

Описано критерії класифікації для систем автоматизованого генерування модульних тестів.

Продемонстровано, що системи генерування модульних тестів є комбінованими та сумісно використовують символьне виконання та оптимізаційні методи для генерування тестів і вхідних даних.

Побудовано універсальну ієрархічну модель системи автоматизованого генерування модульних тестів.

Описано застосування сформованої моделі для підвищення якості та швидкості генерування модульних тестів завдяки новим варіантам декомпозиції задачі на кожному з ієрархічних рівнів.

Вказано на потребу імплементації та подальших обчислювальних експериментів для систем автоматизованого генерування модульних тестів, побудованих на підставі запропонованої моделі.

References

1. Anand, S., Burke, E. K., Chen, T. Y., Clark, J., Cohen, M. B., Grieskamp, W., & Zhu, H. (2013). An orchestrated survey of methodologies for automated software test case generation. *Journal of Systems and Software*, *86*(8), 1978-2001. https://doi.org/10.1016/j.jss.2013.02.061
2. Bjørner, N. (2018). Z3 and SMT in industrial R&D. Springer, Cham. In *International Symposium on Formal Methods*, 675–678. https://doi.org/10.1007/978-3-319-95582-7_44
3. Bucur, S., Ureche, V., Zamfir, C., & Candea, G. (2011). Parallel symbolic execution for automated real-world software testing. *Proceedings of the Sixth Conference on Computer Systems – EuroSys* 11. https://doi.org/10.1145/1966445.1966463
4. Cadar, C., Dunbar, D., & Engler, D. R. (2008). Klee: unassisted and automatic generation of high-coverage tests for complex systems programs. *In OSDI*, Vol. 8, 209–224.
5. Cadar, C., Ganesh, V., Pawlowski, P. M., Dill, D. L., & Engler, D. R. (2008). EXE. *ACM Transactions on Information and System Security*, *12*(2), 1–38. https://doi.org/10.1145/1455518.1455522

6. Campos, J., Panichella, A., & Fraser, G. (2019). EvoSuite at the SBST 2019 Tool Competition. *2019 IEEE/ACM 12th International Workshop on Search-Based Software Testing* (SBST). https://doi.org/10.1109/sbst.2019.00017
7. Chen, J., Hu, W., Zhang, L., Hao, D., Khurshid, S., & Zhang, L. (2018). Learning to accelerate symbolic execution via code transformation. In *32nd European Conference on Object-Oriented Programming* (ECOOP 2018). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.
8. Dutertre, B. (2014). Yices 2.2. Lecture Notes in Computer Science, 737–744. https://doi.org/10.1007/978-3-319-08867-9_49
9. Fraser, G., & Arcuri, A. (2011). EvoSuite. Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering – SIGSOFT/FSE 11. https://doi.org/10.1145/2025113.2025179
10. He, J., Balunović, M., Ambroladze, N., Tsankov, P., & Vechev, M. (2019). Learning to Fuzz from Symbolic Execution with Application to Smart Contracts. *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. https://doi.org/10.1145/3319535.3363230
11. Kaner, C. (1997). The impossibility of complete testing. Software QA. Vol. 4, no. 4, 28 p.
12. Katz, G., Barrett, C., Dill, D. L., Julian, K., & Kochenderfer, M. J. (2017). Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. *Lecture Notes in Computer Science*, 97–117. https://doi.org/10.1007/978-3-319-63387-9_5
13. Lakhotia, K., Harman, M., & Gross, H. (2013). AUSTIN: An open source tool for search based software testing of C programs. *Information and Software Technology*, *55*(1), 112–125. https://doi.org/10.1016/j.infsof.2012.03.009
14. Misailovic, S., Milicevic, A., Petrovic, N., Khurshid, S., & Marinov, D. (2007). Parallel test generation and execution with Korat. In *Proceedings of the the 6th joint meeting of the European software* engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, 135–144. https://doi.org/10.1145/1287624.1287645
15. Poeplau, S., & Francillon, A. (2020). Symbolic execution with SymCC: Dont interpret, compile!. In 2*9th Security Symposium* (Security 20), 181–198.
16. Scalabrino, S., Grano, G., Di, Nucci, D., Guerra, M., De Lucia, A., Gall, H. C., & Oliveto, R. (2018). Ocelot: A search-based test-data generation tool for c. In *2018 33rd IEEE/ACM International Conference on Automated Software Engineering* (*ASE*), 868–871. https://doi.org/10.1145/3238147.3240477
17. Sen, K., Marinov, D., & Agha, G. (2005). CUTE: A concolic unit testing engine for C. *ACM SIGSOFT Software Engineering Notes*, *30*(5), 263–272. https://doi.org/10.21236/ada482657
18. Shiqi, S., Shinde, S., Ramesh, S., Roychoudhury, A., & Saxena, P. (2019). Neuro-Symbolic Execution: Augmenting Symbolic Execution with Neural Constraints. *Proceedings 2019 Network and Distributed System Security Symposium*. https://doi.org/10.14722/ndss.2019.23530
19. Siddiqui, J. H., & Khurshid, S. (2010). ParSym: Parallel symbolic execution. *2010 2nd International Conference on Software Technology and Engineering*. https://doi.org/10.1109/icste.2010.5608866
20. Staats, M., & Păsăreanu, C. (2010). Parallel symbolic execution for structural test generation. *Proceedings of the 19th International Symposium on Software Testing and Analysis* – ISSTA 10. https://doi.org/10.1145/1831708.1831732
21. Wintersteiger, C. M., Hamadi, Y., & de Moura, L. (2009). A Concurrent Portfolio Approach to SMT Solving. *Lecture Notes in Computer Science*, 715–720. https://doi.org/10.1007/978-3-642-02658-4_60
22. Yoshida, H., et. al. (2017). KLOVER: Automatic Test Generation for C and C++ Programs, *Using Symbolic Execution, in IEEE Software*, vol. 34, no. 5, 30–37. https://doi.org/10.1109/MS.2017.3571576

R. P. Bazylevych, A. V. Franko

Lviv Polytechnic National University, Lviv, Ukraine

HIERARCHICAL MODEL OF AUTOMATED TEST GENERATION SYSTEM

The study aims to further enhance the automated unit test generation methods by creating new generalized model of the unit test generation systems that supports metaheuristic and parallel computations. The study uses analysis method to find the current issues of the test generation task, modeling, generalization, and synthesis to propose producing an enhanced model of the test generation systems. A current main method for unit test generation is symbolic execution. Its capabilities are limited by high computation complexity and lack of universality, however many methods exist for boosting the performance in particular cases. The paper classifies current methods based on the input data used and the methods of data processing. The most advanced methods use several sources of input data or several data analysis techniques. As a result of classification, the paper proposes the hierarchical model, which takes into account variations of a symbolic execution unit test generation technique. There are 4 layers in the proposed model. The first layer contains the main algorithm. Consequently, it is responsible for overall process control and test generation. The second layer is a "bridge" between the additional heuristic or artificial intelligence and the main algorithm. Moreover, it combines the output of the third layer (solved logical conditions) with its own higher level rules and heuristic. The third layers consists of SMT (Satisfiability modulo theories)-solvers aimed at solving sets of logical conditions. The fourth layer is the individual algorithms employed by the SMT-solvers. The benefits of proposed model are universality, modularity, and potential to use metaheuristics in every layer of the model. Furthermore, it allows additionally splitting the computation process into layers and performing load balancing on each layer individually. The future work includes building up new systems and creating new metaheuristics based on the proposed model.

Keywords: unit testing; symbolic execution; computation effectiveness; software quality; heuristic algorithms.