DOI: https://doi.org/10.15276/ict.02.2025.06
УДК 004.05:004.42

Формальні методи верифікації програмного забезпечення: сучасні підходи та виклики

Стяглик Наталя Іванівна
Канд. пед. наук, доцент, завідувач кафедри Інформаційних технологій та математичного моделювання
ORCID: https://orcid.org/0000-0002-0573-3508; natalia.stiahlyk@karazin.ua. Scopus ID: 57216944420
Харківський національний університет імені В. Н.Каразіна, майдан Свободи, 4. Харків, 61022, Україна

АНОТАЦІЯ

У дослідженні розглядається проблема підвищення надійності програмного забезпечення за допомогою формальних методів верифікації, які дедалі активніше використовуються в сучасній інженерії програмних систем. На відміну від традиційного тестування, що виявляє лише частину помилок, формальні методи ґрунтуються на математичних моделях і логічних доведеннях, що дозволяє проводити більш повну і достовірну перевірку програмних систем. Актуальність теми зумовлена зростанням складності програмного забезпечення та високими вимогами до його коректності у критичних сферах, таких як авіація, охорона здоров'я, енергетика чи кібербезпека.

В роботі здійснено систематизацію ключових напрямів формальної верифікації. Показано, що формальні специфікації забезпечують точне математичне визначення вимог до програм, виключаючи двозначності природної мови. Модель-чекінг дозволяє автоматизовано досліджувати всі можливі стани системи, виявляючи приховані помилки у поведінці багатопотокових і розподілених програм. Формальне доведення коректності спирається на логічні системи та аксіоматичні методи, що гарантує найвищий рівень впевненості у відповідності програми її специфікації. Символьне виконання відкриває можливості для аналізу всіх шляхів виконання програми й автоматичної генерації тестових даних. Разом із тим окреслено низку викликів, що стримують широке впровадження формальних методів у практику. Серед них – проблема масштабованості, яка обмежує ефективність аналізу великих систем; потреба у висококваліфікованих фахівцях із математичною підготовкою; складність формалізації вимог; труднощі інтеграції у сучасні методології розробки на кшталт Agile та DevOps.

Перспективи розвитку пов'язуються з інтеграцією технологій штучного інтелекту для автоматизації створення специфікацій і побудови доказів, розробкою гібридних методів, що поєднують формальну верифікацію з тестуванням, а також активним застосуванням у сферах кібербезпеки, блокчейн-технологій та автономних транспортних систем. Особливе значення має інтеграція формальних методів у середовища програмування та системи безперервної інтеграції, що підвищує їх доступність для індустрії. Таким чином, формальні методи верифікації залишаються потужним інструментом підвищення надійності програмних систем. Незважаючи на наявні виклики, вони мають значний потенціал для подальшого розвитку та поступового перетворення з академічної дисципліни на невід'ємну складову промислової розробки програмного забезпечення.

Ключові слова: програмне забезпечення; верифікація програмного забезпечення; вимоги до програмного забезпечення; формальні методи; формальні специфікації; доведення коректності; надійність програмних систем

У сучасному світі програмне забезпечення стало фундаментальною основою функціонування більшості сфер людської діяльності – від науки й освіти до промисловості, фінансів, транспорту та охорони здоров'я. Воно забезпечує автоматизацію складних процесів, сприяє підвищенню ефективності управління, відкриває нові можливості для комунікації та обробки даних. З кожним роком масштаби його використання зростають, і суспільство дедалі більше покладається на цифрові рішення у повсякденному житті та професійній діяльності.

Така глобальна інтеграція програмного забезпечення в усі сфери життєдіяльності людини обумовлює необхідність підвищеної уваги до його якості. Обґрунтовується це тим, що на відміну від традиційних матеріальних продуктів, помилки у програмних системах не завжди можна виявити одразу, а їхні наслідки можуть мати накопичувальний або прихований характер. Водночас у світі, де цифрові технології стали основою економіки, безпеки та соціальної взаємодії, ціна навіть однієї критичної помилки зростає у геометричній прогресії. Це підтверджується численними прикладами з практики, коли збої програмного забезпечення призводили до фінансових втрат, зупинки виробничих процесів або навіть створювали загрози для життя і здоров'я людей.

Тому якість і надійність програмного забезпечення є одним із визначальних чинників його конкурентоспроможності та успішного впровадження в різних сферах діяльності. За даними досліджень компанії Consortium for IT Software Quality (CISQ), лише у США економічні втрати від дефектів у програмному забезпеченні щороку сягають понад 2 трлн доларів. Особливо гостро проблема постає у сферах критичного застосування – авіаційній, медичній, енергетичній чи транспортній інфраструктурі, де навіть одна не виявлена помилка може призвести не лише до фінансових збитків, а й до катастрофічних наслідків для безпеки людей і навколишнього середовища.

Традиційні методи тестування, попри свою важливість, мають фундаментальне обмеження – вони досліджують лише частину можливих сценаріїв виконання програми, залишаючи поза увагою значний простір потенційних помилок. У складних програмних системах кількість можливих станів зростає експоненційно, що робить повне тестування практично неможливим. Навіть використання автоматизованих тестових середовищ і генерації випадкових тестів не гарантує охоплення всіх критичних випадків, особливо коли йдеться про системи з високим рівнем паралельності або інтерактивності. Це призводить до того, що деякі помилки виявляються лише на етапі експлуатації, коли їх виправлення коштує значно дорожче, а іноді й неможливе без серйозних наслідків.

Саме тут особливої ваги набувають формальні методи верифікації, які ґрунтуються на математичних моделях, логічних доведеннях та алгоритмічному аналізі поведінки системи. Вони дозволяють не просто перевірити окремі сценарії, а формально довести відповідність програмного забезпечення специфікаціям і вимогам [1]. Такий підхід є особливо актуальним для критично важливих галузей – авіації, автомобільної промисловості, телекомунікацій та медицини, де навіть незначна помилка може спричинити катастрофічні наслідки. Крім того, формальні методи надають можливість виявляти приховані помилки, які неможливо відтворити у звичайних тестах, що робить їх потужним доповненням до традиційних підходів забезпечення якості програмного забезпечення.

Використання формальних методів дозволяє забезпечити доведення коректності програмного забезпечення на рівні строгих математичних гарантій [2]. У світовій практиці вже є приклади їх успішного застосування: перевірка мікропроцесорів (Intel, ARM), сертифікація авіаційних систем відповідно до стандарту DO-178C, розробка безпечних протоколів у блокчейн-технологіях та системах кібербезпеки. Зростання популярності автономних транспортних засобів і “розумних” інфраструктурних систем ще більше актуалізує потребу у використанні таких методів, адже від їхньої надійності залежить життя людей.

Таким чином, формальні методи верифікації стають не лише інструментом підвищення якості програмного забезпечення, а й стратегічним чинником розвитку цифрової економіки та гарантом безпеки у високотехнологічних галузях.

Метою дослідження є аналіз сучасних підходів до формальної верифікації програмного забезпечення та визначення основних викликів їх впровадження. Відповідно, постають завдання: розглянути сутність формальної верифікації програмного забезпечення та її відмінність від традиційних методів тестування; розглянути основні методи формальної верифікації (модель-чекінг, доведення теорем, аналіз абстракцій тощо); визначити ключові переваги та обмеження застосування формальних методів у сучасній інженерії програмного забезпечення; проаналізувати виклики інтеграції формальних методів у процеси Agile та DevOps; окреслити перспективи розвитку формальних підходів та можливі шляхи подолання бар’єрів їх впровадження в практичну діяльність.

Схарактеризуємо теоретичні основи формальних методів. Формальні методи верифікації визначаються як використання строго визначених математичних моделей для опису, аналізу й перевірки властивостей програмних систем. Основними напрямами є: формальні специфікації, що задають поведінку системи; модель-чекінг, що забезпечує автоматизовану перевірку властивостей шляхом дослідження всіх можливих станів моделі; формальне

доведення коректності, яке спирається на побудову математичних доказів; символьне виконання, що дозволяє аналізувати всі можливі шляхи виконання програми.

Розглянемо ці напрямки більш детально.

Формальні специфікації – це математично обґрунтовані описи програмного забезпечення, які задають поведінку системи незалежно від конкретної реалізації. Використання формальних специфікацій дозволяє уникнути притаманних природним мовам двозначностей. Серед найпоширеніших підходів варто відзначити логічні системи (наприклад, темпоральна логіка для опису послідовності подій), алгебраїчні методи (засновані на аксіоматичному задання функцій та операцій), а також онтологічні моделі, що структурують знання у вигляді сутностей та їх відношень. Завдяки формальним специфікаціям можна математично перевірити, чи задовольняє програма заданим вимогам [3].

Модель-чекінг – метод автоматизованої перевірки властивостей програмного забезпечення або апаратних систем на основі аналізу всіх можливих станів їхньої моделі. Ідея полягає в тому, щоб побудувати абстрактну модель системи (наприклад, у вигляді автомата станів) та перевірити, чи виконує вона певні властивості, задані у формальній логіці. Перевагою модель-чекінгу є здатність виявляти приховані помилки, пов’язані з паралельністю або синхронізацією процесів [4]. Серед відомих інструментів – SPIN, NuSMV, UPPAAL. Основним обмеженням цього підходу є так званий “state explosion problem”, коли кількість станів зростає експоненційно зі збільшенням складності системи. Попри це, метод набув широкого поширення у практиці, зокрема в авіаційній, телекомунікаційній та автомобільній галузях, де критично важливим є підтвердження коректності роботи систем.

Використання модель-чекінгу дозволяє на ранніх етапах виявляти помилки, які важко відтворити у звичайному тестуванні, наприклад, умови гонки або тупикові ситуації у багатопотокових програмах. Обґрунтовується його значущість тим, що такі дефекти у реальних умовах експлуатації часто призводять до непередбачуваних і небезпечних наслідків. Для подолання проблеми вибуху станів застосовуються різні стратегії оптимізації, серед яких абстракція моделей, символьна перевірка властивостей за допомогою BDD (Binary Decision Diagrams), а також застосування часткового порядку для зменшення кількості розглянутих міжпроцесних взаємодій. Розвиток цих підходів робить модель-чекінг більш масштабованим і придатним для аналізу сучасних складних систем. А завдяки поєднанню формальної строгості й автоматизації модель-чекінг розглядається як один із найефективніших методів верифікації, який поступово інтегрується в інструментарій інженерії програмного забезпечення поряд з тестуванням, симуляцією та іншими підходами.

Формальне доведення коректності спирається на побудову строгих математичних доказів того, що програма задовольняє свою специфікацію. Використовуються методи логіки вищих порядків, індуктивні доведення та аксіоматичні системи (наприклад, логіка Хоара). Формальне доведення вважається найбільш надійним способом підтвердження правильності роботи програмного забезпечення, однак воно є трудомістким та потребує високої кваліфікації розробників. Інструменти на кшталт Coq, Isabelle/HOL, HOL Light значно полегшують цей процес, дозволяючи автоматизувати частину доведень і створювати бібліотеки вже готових теорем.

Символьне виконання – то метод, який дозволяє проаналізувати всі можливі шляхи виконання програми без необхідності запускати її з конкретними даними. Замість конкретних значень використовуються символічні вирази, які описують множини можливих вхідних даних [5]. У процесі символьного виконання формується система обмежень, яку можна перевірити за допомогою SAT/SMT-розв’язувачів. Це дозволяє виявляти приховані помилки, що проявляються лише за певних комбінацій вхідних даних, а також автоматично генерувати тестові набори. Прикладами інструментів є KLEE для мови C/C++ або Java PathFinder для програм мовою Java.

Аналіз абстракцій є методом формальної верифікації, що ґрунтується на спрощенні програмних моделей задля зручності автоматизованого аналізу їхньої поведінки. Його сутність полягає у заміні конкретних значень змінних та станів на узагальнені характеристики, які відображають лише ті властивості системи, що мають значення для перевірки. Наприклад, замість повного переліку всіх можливих числових значень можна розглядати лише категорії «додатне», «нуль» або «від'ємне». Такий підхід дозволяє досліджувати програму у спрощеному просторі станів, що значно зменшує обчислювальну складність та робить можливим виявлення потенційних помилок, таких як ділення на нуль чи вихід за межі масиву. Особливістю цього методу є його консервативність: якщо в абстрактній моделі виявлено помилку, вона може існувати і в реальній програмі. Водночас така властивість часто породжує хибнопозитивні результати, коли інструмент сигналізує про помилку, якої насправді немає. Ефективність аналізу абстракцій значною мірою залежить від правильного вибору рівня узагальнення. Надмірно грубі абстракції призводять до великої кількості помилкових попереджень, тоді як занадто детальні роблять аналіз повільним і складним. Попри обмеження, метод набув широкого застосування у сфері статичного аналізу коду та верифікації системного програмного забезпечення [6]. Він активно використовується у спеціалізованих інструментах, зокрема у відомому аналізаторі Astrée для програм мовою C, а також інтегрується в компілятори для оптимізації коду й підвищення його безпеки. Завдяки здатності поєднувати формальну строгість із практичною масштабованістю аналіз абстракцій розглядається як один із найважливіших методів формальної верифікації складних програмних систем.

Інтеграція цих методів у сучасні середовища розробки сприяє підвищенню надійності програмного забезпечення, особливо в критичних галузях, зокрема авіаційній, медичній та кібербезпековій.

Проте, попри високий потенціал, застосування формальних методів має низку суттєвих викликів, що обмежують їхнє широке використання в індустрії.

По-перше, масштабованість. Однією з головних проблем є так звана "проблема вибуху станів" (state explosion problem). Для складних програмних або апаратних систем кількість можливих станів зростає експоненційно, що робить їхній повний аналіз практично неможливим. Наприклад, у системах з багатопотоковою обробкою або великою кількістю взаємодіючих модулів кількість комбінацій поведінки настільки велика, що навіть найпотужніші сучасні обчислювальні засоби не здатні їх перевірити. Це призводить до необхідності застосування абстракцій, редукцій чи евристичних методів, які, з одного боку, зменшують складність, а з іншого – можуть знижувати повноту перевірки.

По-друге, потреба у спеціалізованих знаннях. Ефективне застосування формальних методів вимагає від розробників глибоких знань у сфері математичної логіки, теорії автоматів, алгоритмів та спеціалізованих мов специфікацій. Звичайні програмісти, які працюють у рамках прикладних задач, рідко володіють достатньою теоретичною підготовкою для застосування таких інструментів. Це створює бар'єр для їхнього широкого впровадження, оскільки компанії змушені інвестувати значні ресурси у підготовку кадрів або залучення вузьких фахівців.

Крім того, складність формалізації вимог, що є ще одним важливим викликом у практиці розробки програмних застосунків. Вимоги до програмного забезпечення часто формулюються природною мовою, яка сама по собі є неоднозначною та неповною. Перетворення таких специфікацій у формальні моделі вимагає додаткових зусиль, а іноді й глибокої переробки процесів збору вимог. Неповні чи нечіткі специфікації призводять до того, що результати верифікації не повністю відповідають очікуванням, або ж перевірка здійснюється лише для частини поведінки системи. Це суттєво обмежує ефективність формальних методів на практиці.

І, нарешті, інтеграція в життєвий цикл розробки. Сучасні підходи до створення програмного забезпечення, зокрема Agile та DevOps, орієнтовані на швидкі ітерації та безперервну інтеграцію. Формальні методи, які передбачають ретельне створення моделей та

довготривалу перевірку, не завжди легко вписуються у такі процеси. Їх застосування може сповільнювати темпи розробки, що є неприйнятним для багатьох комерційних проєктів, де конкурентна перевага залежить від швидкого виходу продукту на ринок. Окрім цього, більшість інженерів-програмістів не мають достатньої підготовки у сфері формальних методів, що створює додатковий бар'єр для їх широкого впровадження.

Водночас зростає інтерес до поєднання формальних методів із практиками безперервного тестування та автоматизованої верифікації у рамках CI/CD-процесів. Ідея полягає у створенні спрощених або часткових моделей, які можна інтегрувати в окремі етапи розробки без суттєвого уповільнення циклу. Такі підходи дозволяють виявляти критичні помилки на ранніх стадіях і знижувати витрати на їх усунення, не жертвуючи при цьому гнучкістю [7]. Крім того, розвиток інструментів автоматизації та поява гібридних методологій (наприклад, поєднання тестування з модель-чекінгом чи використання контрактного програмування у середовищах Agile) відкривають перспективи гармонійної інтеграції формальних методів у сучасну інженерію програмного забезпечення.

Таким чином, попри наявність низки обмежень, розвиток формальних методів не зупиняється, а сучасні дослідження спрямовані на пошук шляхів подолання виявлених утруднень. Усвідомлення наявних обмежень не лише не зменшує значення формальних методів, а й формує підґрунтя для пошуку інноваційних шляхів їхнього розвитку. Одним із перспективних напрямів є інтеграція технологій штучного інтелекту, які здатні автоматизувати процес формування формальних специфікацій та підтримувати побудову доказів, що традиційно вимагають значних зусиль експертів. Окрему увагу заслуговують гібридні підходи, що поєднують тестування та формальну верифікацію, дозволяючи забезпечити баланс між точністю математичного аналізу та практичною гнучкістю. Сфери кібербезпеки, блокчейн-технологій та автономних транспортних систем стають ключовими галузями для впровадження таких рішень, адже саме тут вартість помилки є критично високою. Важливим є також удосконалення інструментальної підтримки: інтеграція формальних методів у популярні середовища програмування та платформи безперервної інтеграції (CI/CD) робить їх більш доступними для практичного застосування. Сукупність цих тенденцій вказує на те, що у майбутньому формальні методи можуть перестати бути суто академічним інструментом і перетворитися на невід'ємний елемент промислової розробки програмного забезпечення.

Отже, формальні методи верифікації програмного забезпечення є важливим інструментом забезпечення його надійності та безпеки. Незважаючи на виклики, пов'язані зі складністю й потребою у високій кваліфікації, ці методи мають значний потенціал, особливо у критичних сферах застосування. У майбутньому можна очікувати їх тіснішої інтеграції з інтелектуальними технологіями та широкого впровадження в індустрію.

СПИСОК ЛІТЕРАТУРИ

1. Гейко О. О. «Забезпечення правильності комп'ютерних моделей через верифікацію та валідацію». *Вчені записки ТНУ імені В. І. Вернадського. Серія: Технічні науки*. 2023; 34 (73), № 4. С. 30–37. DOI: https://doi.org/10.32782/2663-5941/2023.4/06.

2. "Galois What Are Formal Methods?" URL: https://www.galois.com/what-are-formal-methods#section-benefits-of-formal-methods. 2025.

3. Ter Beek M. H. "Formal Methods in Industry". 2024. DOI: https://doi.org/10.1145/3689374.

4. «Тестування програмного забезпечення : навч. посіб.». *Черкаси : ЧНУ імені Богдана Хмельницького*. 2017. URL: https://eprints.cdu.edu.ua/1482/1/testyvan.pdf.

5. Li Y., Meng R., Duck G. J. "Large Language Model powered Symbolic Execution". 2025. DOI: https://doi.org/10.1145/3763163.

6. Cousot P. "Abstract Interpretation: From 0, 1, To ∞". 2023. URL: https://cs.nyu.edu/~pcousot/publications.www/CSV-2023-cousot.pdf.

7. Чанкветадзе Д. «Методологічні аспекти інтеграції формальної верифікації у CI/CD конвеєри». *Вісник ХНТУ. Інформаційні технології*. 2025; 1 (92): 245–250. DOI: https://doi.org/10.35546/kntu2078-4481.2025.1.2.34.

DOI: https://doi.org/10.15276/ict.02.2025.06
UDC 004.05:004.42

Formal methods of software verification: modern approaches and challenges

Natalia I. Stiahlyk
PhD, Associate Professor, Head of the Department of Information Technologies and Mathematical Modeling
ORCID: https://orcid.org/0000-0002-0573-3508; natalia.stiahlyk@karazin.ua. Scopus ID: 57216944420
V.N. Karazin Kharkiv National University, 4, Svobody Square. Kharkiv, 61022, Ukraine

ABSTRACT

The study examines the problem of increasing software reliability using formal verification methods, which are becoming increasingly common in modern software systems engineering. Unlike traditional testing, which detects only a portion of errors, formal methods rely on mathematical models and logical reasoning, enabling more comprehensive and reliable verification of software systems. The relevance of the topic is driven by the growing complexity of software and the strict requirements for its correctness in critical areas such as aviation, healthcare, energy, and cybersecurity.

The paper systematizes the key directions of formal verification. It demonstrates that formal specifications provide a precise mathematical definition of software requirements, eliminating ambiguities inherent in natural language. Model checking makes it possible to automatically explore all possible system states, revealing hidden defects in the behavior of multi-threaded and distributed programs. Formal proof of correctness is based on logical systems and axiomatic methods, guaranteeing the highest level of confidence that the program meets its specification. Symbolic execution enables analysis of all execution paths and automatic generation of test data. At the same time, a number of challenges that hinder the widespread adoption of formal methods in practice are identified. These include the scalability problem, which limits the efficiency of analyzing large systems; the need for highly qualified specialists with a strong mathematical background; the complexity of formalizing requirements; and the difficulties of integrating these methods into modern development methodologies such as Agile and DevOps.

Future prospects are associated with the integration of artificial intelligence technologies to automate the creation of specifications and the construction of proofs, the development of hybrid methods combining formal verification with testing, and the active use of these approaches in cybersecurity, blockchain technologies, and autonomous transport systems. Of particular importance is the integration of formal methods into programming environments and continuous integration systems, which increases their accessibility for industry. Thus, formal verification methods remain a powerful tool for improving software system reliability. Despite the existing challenges, they hold significant potential for further development and for gradually evolving from an academic discipline into an essential component of industrial software development.

Keywords: Software; software verification; software requirements; formal methods; formal specifications; proof of correctness; software system reliability