Стаття містить опис методів статичної та динамічної верифікації послідовно-паралельного програмного коду на моделі у вигляді управляючої мережі Петрі зі стохастичними елементами, яка будується за деревом розбору. Стаття містить приклад практичної реалізації розглянутих методів

Ключові слова: динамічна верифікація, дерево розбору, мережа Петрі, стохастичні елементи

Статья содержит описание методов статической и динамической верификации последовательно-параллельного программного кода на модели в виде управляющей сети Петри со стохастическими элементами, которая строиться по дереву разбора. Статья содержит пример практической реализации рассмотренных методов

Ключевые слова: динамическая верификация, дерево разбора, сеть Петри, стохастические элементы

УДК 004.415.52, 004.052.42

МЕТОДИ ДИНАМІЧНОЇ ВЕРИФІКАЦІЇ ПРОГРАМНОГО КОДУ НА ОСНОВІ ІНТЕРПРЕТАЦІЙ МЕРЕЖ ПЕТРІ

Б.А. Гнесюк*
E-mail: psilon_vlksm@mail.ru
О.О. Супруненко
Кандидат технічних наук, доцент*
Контактний тел.: 066-187-99-50
E-mail: ra-oks@mail.ru
*Кафедра програмного забезпечення автоматизованих систем
Черкаський національний університет імені Богдана Хмельницького
бул. Шевченка, 81, м. Черкаси, Україна, 18031

1. Вступ

Розвиток багатопроцесорних та багатоядерних комп'ютерних систем потребує створення програмного коду з паралельними ділянками, що дозволяє значно підвищити ефективність обробки великих обсягів інформації, а з іншого боку породжує проблему перевірки паралельного програмного коду, яка є значно складнішою ніж у випадку перевірки послідовних програм.

Однією з актуальних проблем підвищення якості розробки програмних систем, особливо систем з кодом, що містить директиви розпаралелювання та паралельно виконувані ділянки коду, є запровадження у процес розробки верифікації програмної моделі. Верифікація [1, 2] спрямована на аналіз моделі програмного коду щодо відповідності технічним вимогам та виявлення дефектів та/або умов їх виникнення у програмному коді. Верифікація проводиться на моделях [1], що описують специфіку функціонування програмної системи при різних вхідних даних та різних часових і комунікаційних умовах виконання послідовності інструкцій.

Розглядаючи верифікацію програмного коду, потрібно передбачити побудову та використання моделі, в яку адекватно перетворюється код, а також яка дозволить проводити аналіз на основі характерних критичних ситуацій.

2. Виділення проблеми, постановка задачі

Верифікація коду передбачає формальну інспекцію програмного коду на адекватній моделі з метою зменшення часу тестування коду без зниження його якості [2]. На відміну від відладки, яка передбачає локалізацію та усунення помилок, верифікація спрямована на демонстрацію наявності помилок та умов їх виникнення, а також є контрольованим і керованим процесом, який дозволяє також проаналізувати наслідки виправлення виявлених помилок. Верифікація фактично є процесом забезпечення заздалегідь визначеного рівня якості програмного продукту [2].

Для верифікації застосовують статичні та динамічні методи. Статичні методи полягають в аналізі моделі під час побудови та перед запуском на імітацію, а динамічні – при імітуванні функціонування моделі. Серед методів статичного аналізу розглядають метод Data-flow Analysis (аналіз потоку даних) [3], який передбачає аналіз можливих наборів значень змінних, що розраховані в різних точках графу потоку управління цієї програми. Він використовується для знаходження помилок в логіці програми типу зациклень, неініціалізованих змінних, помилок в математичних виразах, але майже не дає можливості виявляти помилки в паралельному коді.

Метод абстрактної інтерпретації [4] базується на математичних теоремах, що визначаються правила для аналізу складних динамічних систем, якими є і програмні додатки. Множину станів програми представляють у абстрактному вигляді та визначають правила роботи з ними. Далі проводиться інтерпретація побудованої моделі, яка є досить великою задачею (аналізує еволюцію всіх змінних коду), але може виконуватися на потужних персональних комп'ютерах за допустимий для аналізу час. Метод абстрактної інтерпретації дозволяє виявити неправильні перетворення типів; виходи за межі масиву або типу; помилки в математичних виразах; код, що ніколи не виконається та ін., але потребує досить кропіткої роботи по формуванню абстрактної моделі, та чималого часу й ресурсів для інтерпретації моделі.

Метод Model Checking проводить перевірку програми на формальній моделі, що побудована на основі темпоральних логік [1]. Даним методом вдається виявити широкий спектр помилок та потенційних умов їх виникнення в моделях функціонування динамічних систем, але для цього потрібно побудувати адекватну модель системи з кінцевим числом станів [1], що є досить складною задачею.

Code Coverage Analysis (аналіз покриття коду) проводить динамічний аналіз програмного коду шляхом його багаторазового запуску з різними вхідними даними, що дозволяють забезпечити максимальне покриття коду. На практиці вдається досягти до 80% покриття коду, але код, що ніколи не виконається, не може бути визначений цим методом [5]. Частина коду, що непокрита перевірками не може бути проконтрольована, що дозволяє дуже наближено визначити рівень надійності програми.

Метод Program Slicing використовується як допоміжний для динамічної верифікації, що дозволяє за спеціальними алгоритмами відсікати не суттєві ділянки коду [6]. Однак раніше відсічені ділянки можуть бути перевірені при наступних запусках програми з іншими вхідними даними. Неточність алгоритмів відсікання приводять до пропуску помилок, що не дозволяє використовувати цей алгоритм самостійно.

Метод Assertions (перевірка твердження) полягає в тому, що програміст під час написання програми в критичних місцях робить твердження про значення змінних в цьому місці [7]. При динамічному тестуванні ці твердження перевіряються на правдивість. Якщо якесь з стверджень виявилося хибним, то це вказує на помилку. Даний метод призначений для виявлення логічних помилок та помилок обчислень [7-8], але він показує низьку ефективність при помилках розпаралелювання та витіках пам'яті.

Виходячи з огляду методів верифікації проблемою у процесі контролю якості паралельно-послідовних програм є виявлення всіх типів помилок. Постає задача розробки програмного засобу, що сполучує методи статичної та динамічної верифікації, які дозволять виявляти максимальну кількість помилок або потенційні можливості їх виникнення, що характерні для послідовно-паралельних програм. У даній роботі стоять задачі вдосконалення методу динамічної верифікації та побудови моделі, яка дозволяє застосовувати методи статичної та динамічної верифікації до аналізу послідовних і паралельних ділянок програмного коду з метою виявлення дефектів і локалізації ділянок дефектів у програмному коді.

Базовою моделлю для верифікації програмного коду слугує повне дерево розбору [2], яке дозволяє побудувати відповідну управляючу мережу Петрі [9]. Дана модель дозволяє проводити статичну інспекцію коду та перевіряти динамічні властивості мереж Петрі – живість, безпечність, та неконфліктність, тобто запобігати ряду характерних помилок: взаємоблокування, переповнення, зациклення, появі неініціалізованих змінних та ін.

3. Узагальнена модель верифікації програмного коду

При формуванні узагальненої моделі для верифікації послідовно-паралельного програмного коду (рис. 1) використовується дерево розбору, елементами якого є лінійні ділянки коду, розгалуження, паралельні взаємопов'язані ділянки, цикли. На вхід моделі передається програмний код, що потенційно містить помилки. Метою функціонування моделі є стійкий програмний код, якість якого визначається часом його функціонування без збоїв.

На основі початкового представлення будується модель в термінах безпечної управляючої мережі Петрі зі стохастичними елементами [9]. Стохастичні елементи моделюють випадковий (в деяких межах) час відпрацювання певних інструкцій, які в мережах Петрі представлені вершинами переходів.

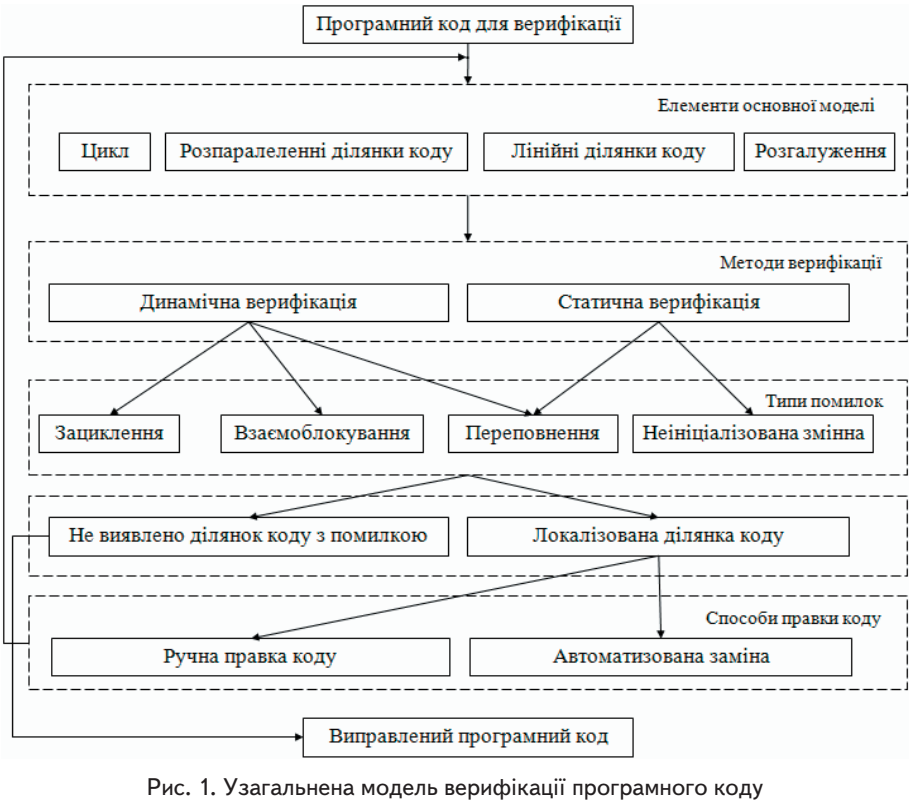


Рис. 1. Узагальнена модель верифікації програмного коду

Модель програмного коду розбивається на ділянки на основі аналізу структури мережі Петрі і змісту окремих алгоритмічних елементів – кожна ділянка вміщує певну алгоритмічну конструкцію, або елемент з паралельними потоками. Лінійні ділянки представляються автоматною мережею Петрі. Окремі ділянки цієї мережі аналізуються за допомогою методів статичної та динамічної верифікації. В результаті можуть

бути виявлені помилки чи потенційні можливості виникнення помилок в коді та визначений їх тип.

За визначеними реальними чи потенційними помилками локалізуються окремі ділянки коду, що їх містять. Знайдені дефекти виправляються в ручному або автоматичному режимі і код перевіряється знову. Таке рішення дозволяє реалізувати функцію контролю за наслідками виявлених помилок.

Коли на черговій ітерації при перегляді робочого коду не виявлено дефектів, на виході моделі формується перевірений код програми. Такий код вважається виправленим (рис. 1).

4. Реалізація моделі верифікації програмного коду в середовищі для перевірки коду на мові програмування C++

Побудована модель була реалізована при створенні середовища верифікації програмного коду на мові програмування C++. Структура середовища відтворює структуру моделі.

Перевірка реалізованих та потенційних помилок відбувається на основі бібліотеки, що включає наступні типи помилок: код, що ніколи не викликається, зациклення, взаємоблокування, некоректна робота з м'ютексом, вихід за межі масиву, розіменування нульових вказівників та ін. Результат роботи верифікатора можна бачити безпосередньо у програмному коді.

Наприклад, у код, що перевіряється, було додано функцію, яка не викликалася в основному коді. При перевірці на моделі, що представлена елементами мережі Петрі (PN), зазначена ділянка буде виглядати, як показано на рис. 2, а у програмному коді – як показано на рис. 3.

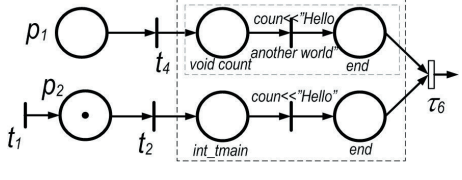


Рис. 2. Мережа Петрі для функції, що має дві незалежні ділянки коду, одна з яких ніколи не виконується (виділена сірим штрихпунктиром)

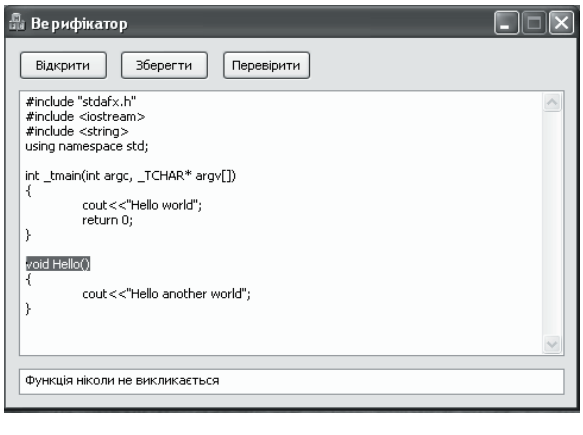


Рис. 3. Виділення у програмному коді ділянки коду, яка ніколи не виконується

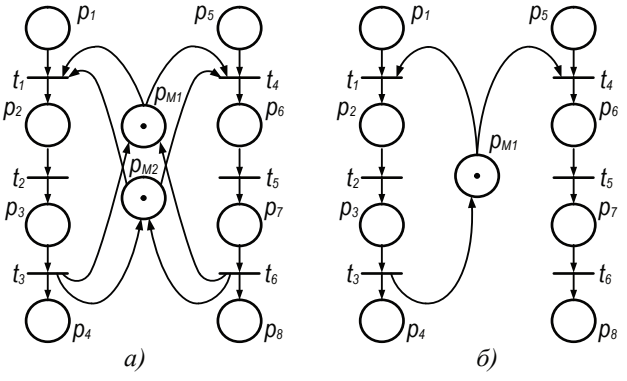
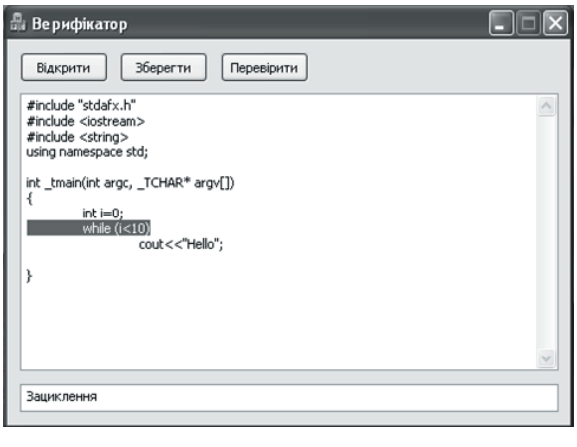


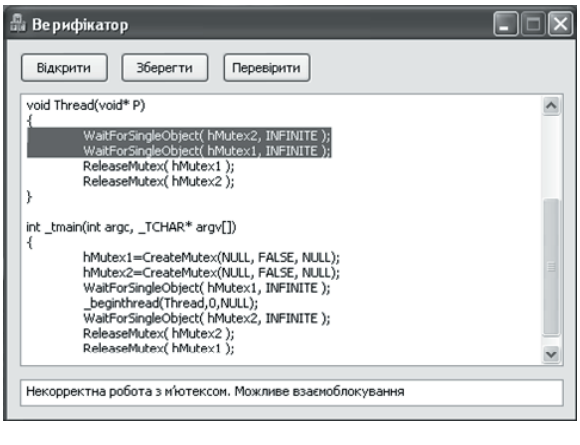
Рис. 4. Моделі м'ютексів з взаємоблокуванням (а) та неповнотою зв'язків (б)

На рис. 4а подано PN-модель з взаємоблокуванням, на рис. 4б – модель м'ютексу з неповною реалізацією зв'язків. На рис. 5а представлено виділення у верифікаторі помилки зациклення. При перегляді коду можна переконатися, що цикл while (i<0) ніколи не завершиться. На рис. 4б показано виявлення можливого взаємоблокування.

Верифікатор не тільки виявляє взаємоблокування, але й підсвічує ресурси, які викликали дану помилку.



а)



б)

Рис. 5. Виділення у програмному коді: а – зациклення, б – некоректної роботи з м'ютексом

5. Висновки

У статті викладений опис методів статичної та динамічної верифікації програм. Проблема дослідження полягає у розширенні набору виявлених помилок в послідовно-паралельному програмному коді, що не забезпечується жодним з окремих методів верифікації. Тому для розв'язання даної проблеми у роботі запропоновано побудувати модель, що використовує методи статичної та динамічної верифікації, метод абстрактної інтерпретації та Model Checking. Метод динамічної верифікації вдосконалений за рахунок застосування критичних властивостей управляючих мереж Петрі. Задача побудови адекватної моделі вирішена шляхом формування остаточної моделі на основі проміжної – повного дерева розбору, та представлення остаточної моделі в термінах мереж Петрі зі стохастичними елементами. У статті представлена програмна реалізація моделі у вигляді засобу верифікації, що дозволяє перевіряти послідовно-паралельний код програм, написаних на мові програмування C++.

Література

1. Карпов, Ю.Г. MODEL CHECKING. Верификация параллельных и распределенных программных систем [Текст] / Ю.Г. Карпов. – СПб.: БХВ-Петербург, 2010. – 560 с.
2. Синицын, С.В. Верификация программного обеспечения (ISBN: 978-5-94774-825-3) [Электронный документ] / С.В. Синицын, Н.Ю. Налютин. Режим доступа: http://www.intuit.ru/department/se/verify/. Проверено 20.12.12.
3. Data-flow analysis. Available at: http://en.wikipedia.org/wiki/Data-flow_analysis. (accessed 15 January 2013).
4. Верификация кода и обнаружение ошибок исполнения путем абстрактной интерпретации. [Электронный документ]. Режим доступа: http://sl-matlab.ru/news/detail.php?ID=723. Проверено 17.12.2012.
5. Steve Cornett. Code Coverage Analysis. Available at: http://www.bullseye.com/coverage.html (accessed 7 December 2012).
6. Frank Tip. A survey of program slicing techniques. Journal of Programming Languages, Volume 3, Issue 3, pages 121–189, September 1995.
7. Programming With Assertions. Available at: http://docs.oracle.com/javase/ 1.4.2/docs/guide/lang/assert.html (accessed 9 December 2012).
8. Rob Smith. J2SE 1.4 Assertion Facility. Available at: http://jnb.ociweb.com/jnb/jnbApr2002.html (accessed 9 December 2012).
9. Кузьмук, В.В. Модифицированные сети Петри и устройства моделирования параллельных процессов: Монография [Текст] / В.В. Кузьмук О.О. Супруненко. – К.: Маклаут, 2010. – 252 с.

Abstract

One of the topical problems in the development of secure applications is the improvement of the quality of the software systems with parallel-sequential code. The verification of a software project is able to solve this problem, subject to the construction of adequate models of functioning of applications and the use of qualitative methods of verification of parallel-sequential programming models. The article considers the methods of static and dynamic verification of applications. The structure of the model for verification was proposed to be build on the basis of an intermediate model that is a complete parse tree. The working model was formed as the control Petri net (SN) with stochastic elements that allowed simulating the functioning of application under different time and communication conditions, and allowed static and dynamic analysis of critical situations. While constructing the model we have implemented the control function of the verifier, which consisted in analysis of the effects of correction of defects in the code, which would help achieve the control of the verification of applications. The article contains the example of the practical implementation of these methods - a verification of the applications written in C ++

Keywords*: dynamic verification, parse tree, Petri net, stochastic elements*