КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Кафедра прикладної статистики

Кваліфікаційна робота бакалавра

на тему:

ДОВЕДЕННЯ КОРЕКТНОСТІ ПАРАЛЕЛЬНИХ ПРОГРАМ

Студента 4 курсу

Жигалла Андрія Андрійовича

Науковий керівник:

доцент, кандидат фіз.-мат. наук

Панченко Т.В.

Робота заслухана на засіданні кафедри прикладної статистики та рекомендована до захисту в ДЕК, протокол № 10 від 01 червня 2016 р.

Завідувач кафедри прикладної статистики

професор, доктор фіз.-мат. наук Лебєдєв Є.О.

Київ – 2016

ЗМІСТ

АНОТАЦІЇ

Дипломна робота складається зі вступу, 2 розділів, висновків, списку використаних джерел (58 найменувань). Загальний обсяг роботи становить 50 сторінок, основний текст роботи викладено на 35 сторінках.

Ключові слова: алгоритм Пітерсона, взаємне виключення, тотальна коректність програм, формальна верифікація, паралельна програма, interleaving, IPCL, композиційно-номінативні мови.

Реферат. В роботі доведено тотальну коректність алгоритму Пітерсона. За програмою зафіксовано стани та переходи транзиційної системи. Середовище виконання – паралельне з почерговим переключенням зі спільною пам'яттю. Сформульовано інваріант. Судження проведено в рамках методу доведення властивостей програм в Interleaving Parallel Compositional Languages (IPCL) – композиційні мови з паралелізмом в режимі чергування. Зроблено висновки щодо адекватності застосування методу для подібних задач завдяки гнучкості композиційно-номінативної платформи та його практичності і легкості застосування для реальних систем.

Key Words: Peterson's algorithm, mutual exclusion, software total correctness, formal verification, concurrent program, interleaving, IPCL, composition-nominative languages.

Abstract. The total correctness of the Peterson's Algorithm has been proved. States and transitions were fixed by the program. Runtime environment considered is interleaving concurrency with shared memory. Invariant of the program was constructed. All reasoning provided in terms of Method for software properties proof in Interleaving Parallel Compositional Languages (IPCL). Conclusions about adequacy of the Method usage for such a kind of tasks (thanks to flexibility of composition-nominative platform) and its practicality as well as ease of use for real-world systems have been made.

ВСТУП

Майже щодня, оперуючи сучасними комп’ютерами, чи працюючи з новітнім програмним забезпеченням, нам доводиться мати справу з системами, які використовують взаємодію через спільну пам'ять [1]. Це - суперкомп’ютери, операційні системи, системи управління базами даних, системи з централізованим сховищем даних, служби операційних систем, сервери в клієнт-серверних середовищах тощо. Разом з цим, сучасна ситуація з технічним забезпеченням, а саме з процесорами, вказує на тенденцію до нарощування обчислювальної потужності за рахунок нарощування кількості ядер процесору, що призводить до розвитку паралелізму в програмному забезпеченні. Таким чином, в наші дні паралельне програмування набуває все більшого поширення, а питання розробки формального методу для доведення коректності систем в перерахованих середовищах залишається актуальним.

Існує ряд підходів до вирішення цієї задачі, але більшість з них мають суттєві недоліки при їх практичному застосуванні. Ці підходи є дуже складними, чи занадто теоретизованими, а деякі з них неможливо застосувати для вирішення реальних задач взагалі. Для прикладу, метод Овіцкі-Гріса [2] вимагає квадратичного числа верифікацій у відповідності до кількості операторів в програмі. Розширена версія rely-guarantee методу Овіцкі-Гріса [3] потребує введення додаткових змінних, а також неочевидного формулювання rely- та guarantee- умов для розв’язання поставленої задачі. В TLA [4] (Temporal Logic of Actions) Лемпорт пропонує створення моделі, яка не є набагато простішою за дві попередні. Більше того, TLA характеризується великим розривом між програмою і її формулою доведення. Таким чином IPCL [5] (Interleaving Parallel Composition Language) - композиційна мова з паралелізмом в режимі чергування може стати одним з найбільш ефективних рішень.

Об’єктом досліджень в даній роботі є доведення властивостей паралельних програм, предмет досліджень – метод доведення властивостей у IPCL та його застосування.

Метою даної дипломної роботи є доведення коректності алгоритму Пітерсона за допомогою методу для доведення програмних властивостей в IPCL.

Новизна полягає у застосуванні новітнього методу для доведення властивостей паралельних програм.

Практична значущість роботи полягає у застосуванні методу до прикладних комерційних програмних систем.

РОЗДІЛ 1
КОМПОЗИЦІЙНО-НОМІНАТИВНІ МЕТОДИ ВЕРИФІКАЦІЇ. КОРЕКТНІСТЬ СЕРВЕРНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Розробка програмного забезпечення, коректного відносно його специфікацій

Формальні методи – техніки, що ґрунтуються на математиці, для специфікації, розробки та верифікації програмних та апаратних систем [9,10].

В інженерній практиці, предикати мають додаткову роль специфікацій, які визначають мету продукту, описуючи необхідні властивості системи, що не існує в реальному світі, але може бути побудована [8]. Предикат, який використовуються як специфікація, повинен описувати бажану систему якомога ясніше і чіткіше.

Програмна специфікація – точний опис того результату, який необхідно досягти за допомогою програми. Цей опис повинен точно встановлювати, що повинна робити програма, не вказуючи, як вона повинна це робити [11].

Верифікація програми – метод, який переконує в тому, що програма буде виконувати саме те, що від неї очікується [11].

1.1.1 Перевірка коректності програмного забезпечення: тестування та верифікація

На сьогодні існує багато методів, що надають можливість пересвідчитися в коректності роботи розробленого (тобто, вже готового) програмного забезпечення. Найчастіше, це – методи тестування програмного забезпечення.

В програмній інженерії довге експериментування з програмою є звичайним методом для визначення її властивостей, що називається тестуванням програми, натомість головною метою наукових теоретичних досліджень є заміна масивного тестування математичним численням, яке базується на теорії та

дозволяє перевіркою лише кількох ключових властивостей (формально) встановити коректність програми [8].

Дейкстра в [13] вказав на головне обмеження тестування – воно може виявити наявність помилок, але не може продемонструвати їх відсутність. Адже з того, що система успішно пройшла тестування на різних рівнях (помодульно та в цілому), ще не обов'язково випливає, що система не містить помилок.

1.1.2 Використання формалізації процесу для розробки програмних систем

До розробки програмного забезпечення можна підходити і з іншого боку, використовуючи методики, що підтримують весь життєвий цикл програм, тобто надають комплексний підхід до розробки програмних систем. Це, наприклад, метод Unified Modeling Language (UML) [14] (див. також [12]), в основу якого покладено парадигму об'єктного підходу.

Існує низка ґрунтовних методів, що також підтримують весь цикл розробки програмних систем та забезпечують такий процес додатковими суттєвими властивостями (такими як поетапність, верифікованість, підконтрольність, строгість [15]) [16]. Це, наприклад, мова формальних специфікацій Meta-IV та метод VDM [17,18], метод формальних специфікацій RAISE [15,19] – як подальший розвиток цього методу, Z [25,26], B [27,26] та інші.

До методів розробки програмного забезпечення "згори донизу", що відповідає природній стратегії людського мислення при побудові систем, можна віднести такі, що підтримують композиційний стиль та композиційну структуру програм. Зараз ці методи набувають розповсюдження. Серед них можна виділити: композиційне програмування [21-24], експлікативне програмування [28], еталонне програмування [29], композиційно-номінативне програмування [30,20,10].

Всі перелічені підходи є варіантами формальних методів. Щодо питання порівняння різних формальних методів, то воно, як правило, досліджується в

контексті вирішення певної задачі, або стосовно деякого кола властивостей – див., наприклад, [50-54].

1.1.3 Формальні методи на практиці

Кріс Джордж, один з розробників методу RAISE та мови формальних специфікацій RSL, в [15] зазначає, що, хоча конструювання систем на базі комп'ютерів є відносно молодою професією, як і в будь-якій галузі інженерії, можна констатувати факт, що побудова великих і складних систем є нелегкою задачею і вимагає використання додаткового набору деяких технік (методик).

Математичні техніки для специфікацій, розробки та верифікації систем програмного забезпечення, що часто називаються формальними методами, дедалі більше застосовуються в індустрії для конструювання реальних систем. Формальні методи вже не є лише предметом теоретичного вивчення, вони все більше використовуються у виробництві (в індустрії). По-перше, підходящі зрілі теорії і формалізми на даний момент перетворилися в практичні методики та техніки. По-друге, зростає число професіоналів, обізнаних з такими техніками. По-третє, зростає наполеглива вимога використання таких методик, особливо для розробки систем, функціонування або несправна робота яких може суттєво вплинути на життя, власність або суспільство [15] (тобто коли збої в роботі програмного забезпечення призводять до втрати життя або власності).

1.2 Методи доведення коректності програмного забезпечення

В галузі програмного забезпечення існують дві класичні проблеми: це побудова програмного забезпечення, яке відповідає специфікації, тобто початковим вимогам – проблема синтезу, та визначення, чи відповідає деяка програма певним вимогам – аналіз.

В контексті доведення коректності програмного забезпечення слід зазначити, що вирішення проблеми синтезу позбавляє необхідності розв'язання

проблеми аналізу, але лише частково. Так, оскільки синтезована програма відповідає початковим вимогам, то для отримання коректного програмного забезпечення необхідно лише правильно сформулювати ці початкові вимоги. Тоді ніяких доведень більше не потрібно. Цей напрямок, дуальний до доведення властивостей програмного забезпечення, активно розвивається, зокрема, Чеботаревим А. Н. в Інституті кібернетики ім. В. М. Глушкова НАНУ [31-34].

Але частковість вирішення питання полягає у тому, що ми вже маємо багато розробленого програмного забезпечення (наприклад, так зване legacy software), яке не було побудовано в свій час за такими принципами. Для таких програм потрібно або створити їх наново згідно відповідних принципів синтезу, або ж для них залишається відкритим питання доведення коректності – проблема аналізу.

Таким чином, проблема доведення коректності програмного забезпечення досі залишається актуальною.

1.2.1 Послідовні програми

Першими роботами, які внесли суттєве розуміння в питання доведення коректності програм, були праці Флойда (Floyd) [35] та Хоара (Hoare) [36]. Флойд ввів сучасне поняття коректності для послідовних програм: часткова коректність, визначена перед- та постумовами, доповнена зупинкою програми. Згідно з методом Флойда часткова коректність доводиться шляхом анотації кожної контрольної точки твердженням, яке повинно бути істиною, коли управління передається (або потрапляє) в цю точку. Доведення розпадається на окремі перевірки таких умов для кожного оператору програми. Зупинка програми доводиться “спадаючими аргументами”: для кожного циклу обирається функція, значення якої є натуральним числом, що зменшується з кожною ітерацією циклу.

Хоар, в свою чергу, переробив метод Флойда для доведення часткової коректності в логічну теорію. В логіці Хоара формули мали вигляд P{S}Q і

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Панченко Т.В. Композиційні методи специфікації та верифікації програмних систем. Дисертація на здобуття наукового ступеня кандидата фізико-математичних наук.: 01.05.03 / Т.В. Панченко – Київ, 2006. – 177 с.
2. Owicki S. An Axiomatic Proof Technique for Parallel Programs / S. Owicki, D. Gries // Acta Informatica. – 1976. – Vol. 6, № 4. – P. 319–340.
3. Xu Q. The Rely-Guarantee Method for Verifying Shared Variable Concurrent Programs / Q. Xu, W.-P. de Roever, J. He // Formal Aspects of Computing. – 1997. – Vol. 9, № 2. – P. 149–174.
4. Lamport L. Verification and Specification of Concurrent Programs / L. Lamport // deBakker J., deRoever W., Rozenberg G. (eds.) A Decade of Concurrency, Vol. 803. – Berlin: Springer-Verlag, 1993. – P. 347–374.
5. Панченко Т.В. Методологія доведення властивостей програм в композиційних мовах IPCL / Т.В. Панченко // Доповіді Міжнарод-ної конференції “Теоретичні та прикладні аспекти побудови програмних систем” (TAAPSD’2004). – К., 2004. – С. 62–67.
6. G. L. Peterson: "Myths About the Mutual Exclusion Problem", Information Processing Letters 12(3) 1981, 115–116.
7. Панченко Т.В. Метод доведення властивостей програм в композиційно-номінативних мовах IPCL / Т.В. Панченко // Проблеми програмування. – 2008. – №1. – С. 3–16.
8. Hoare C.A.R., He J. Unifying Theories of Programming. – London: Prentice Hall Europe, 1998. – 298 p.
9. WikiPedia,Інтернет-енциклопедія. (http://en.wikipedia.org/wiki/Formal_methods)
10. Басараб И.А., Никитченко Н.С., Редько В.Н. Композиционные базы данных. – К.: Либідь, 1992. – 191 с.

11. Толковый словарь по вычислительным системам/Под ред. В. Иллингуорта и др.: Пер. с англ. А. К. Белоцкого и др.; Под ред. Е. К. Масловского. – М.: Машиностроение, 1991. – 560 с.: ил.
12. Бабенко Л.П., Лавріщева К.М. Основи програмної інженерії: Навч. посіб. – К.: Т-во "Знання", 2001. – 269 с.
13. Dijkstra E.W. Why Correctness must be a Mathematical Concern // Boyer R.S., Moore J.S., (eds.), The Correctness Problem in Computer Science. – London: Academic Press, 1981. – P. 1–8.
14. Rumbaugh J., Jacobson I., Booch G. The Unified Modeling Language Reference Manual. – Addison Wesley Longman, Inc., 1999. – 550 p.
15. The RAISE Language Group. The RAISE Specification Language. BCS Practitioner Series. – Prentice Hall, 1992. – 397 p.
16. Панченко Т.В. Використання формальних специфікацій для розробки програмних систем // Вісник Київського університету. Серія: фіз.-мат. науки. – 2002. – Вип. 2. – С. 245–256.
17. Bjørner D., Jones C.B. The Vienna Development Method: The Meta-Language. – Vol. 61 of Lecture Notes in Computer Science. – Springer-Verlag, Heidelberg, Germany, 1978. – 215 p.
18. Bjørner D., Jones C.B. Formal Specification and Software Development. – Prentice-Hall, 1982. – 368 p.
19. The RAISE Method Group. The RAISE Development Method. BCS Practitioner Series. – Prentice Hall, 1995. – 493 p.
20. Никитченко Н.С. Композиционно-номинативный подход к уточнению понятия программы // Проблемы программирования. – 1999. – № 1. – С. 16–31.
21. Редько В. Н. Композиции программ и композиционное программирование // Программирование. – 1978. – № 5. – С. 3–24.
22. Редько В.Н. Основания композиционного программирования // Программирование. – 1979. – № 3. – С. 3–13.

23. Редько В.Н. Композиционная структура программологии // Кибернетика и системный анализ. – 1998. – № 4. – С. 47–66.
24. Редько В.Н. Основания программологии // Кибернетика и системный анализ. – 2000. – № 1. – С. 3–27.
25. Woodcock J.C.P., Davies J. Using Z: Specification, Refinement and Proof. – Prentice Hall, 1996. – 523 p.
26. Abrial J.-R., Schuman S.A., Meyer B. Specification Language Z. – Boston: Massachusetts Computer Associates Inc., 1979. – 378 p.
27. Abrial J.-R. The B-Book: Assigning Programs to Meanings. – Cambridge University Press, 1996. – 779 p.
28. Редько В.Н. Экспликативное программирование: ретроспективы и перспективы // Перша міжнародна науково-практична конференція з програмування УкрПРОГ'98: Праці. – Київ, Кібернетичний центр НАН України. – 1998. – С. 22–41.
29. Редько В.Н., Гришко Н.В., Редько И.В. Эталонное программирование: ретроспективы и перспективы // Тр. 2-й Международной научной-практической конференции по программированию. – Киев: Кибцентр НАНУ, 2000. – С. 13–30.
30. Nikitchenko N. A Composition Nominative Approach to Program Semantics. – Technical Report IT-TR: 1998-020. – Technical University of Denmark. – 1998. – 103 p.
31. Чеботарев А.Н. Синтез процедурного представления автомата, специфицированного в логическом языке *L**. I // Кибернетика. – 1997. – № 4. – С. 60–74.
32. Чеботарев А.Н. Синтез процедурного представления автомата, специфицированного в логическом языке *L**. II // Кибернетика. – 1997. – № 6. – С. 115–126.
33. Чеботарев А.Н. Теоретико-автоматный подход к верификации реактивных систем // Кибернетика и системный анализ. – 2001. – № 6. – С. 37–49.

34. Chebotarev A.N. Provably-correct development of reactive systems // Proc. Int. Workshop “Reactive techniques and efficient theorem proving” (Kyiv, May 19-31, 2000). – Киев: Ин-т кибернетики НАНУ, 2000. – P. 117–133.
35. Floyd R.W. Assigning Meanings to Programs // Proc. Symposium on Applied Mathematics, Vol. 19: Mathematical Aspects of Computer Science. – American Mathematical Society, 1967. – P. 19–32.
36. Hoare C.A.R. An Axiomatic Basis for Computer Programming // Communications of the ACM. – 1969. – Vol. 12, № 10. – P. 576–583.
37. Lamport L. Verification and Specification of Concurrent Programs // deBakker J., deRoever W., Rozenberg G. (eds.) A Decade of Concurrency, Vol. 803. – Berlin: Springer-Verlag, 1993. – P. 347–374.
38. Ashcroft E.A. Proving assertions about parallel programs // Journal of Computer and System Sciences. – 1975. – № 10. – P. 110–135.
39. Hoare C.A.R. Communicating Sequential Processes. – Prentice Hall International, 1985. – 238 p.
40. Jones C.B. Tentative Steps Toward a Development Method for Interfering Programs // ACM Transactions on Programming Languages and Systems. – 1983. – Vol. 5, № 4. – P. 596–619.
41. Кривий С.Л., Матвєєва Л.Е. Формальні методи аналізу властивостей систем // Кибернетика и системный анализ. – 2003. – № 2. – С. 15–36.
42. Owicki S. and Gries D. An Axiomatic Proof Technique for Parallel Programs // Acta Informatica. – 1976. – Vol. 6, № 4. – P. 319–340.
43. Dijkstra E.W. A personal summary of the Gries-Owicki theory // Dijkstra E.W. (ed.) Selected Writings on Computing: A Personal Perspective. – Springer-Verlag, NewYork, Heidelberg, Berlin, 1982. – P. 188–199.
44. Lamport L. Control predicates are better than dummy variables for reasoning about program control // ACM Transactions on Programming Languages and Systems (TOPLAS). – 1988. – Vol. 10, № 2. – P. 267 – 281.

45. Jones C.B. Development Methods for Computer Programs Including a Notion of Interference: DPhil. Thesis. – Oxford University Computing Laboratory, 1981. – 315 p.
46. Jones C.B. Specification and Design of (Parallel) Programs // Information Processing Letters: IFIP Information Processing'83 (In IFIP 9th World Congress). – 1983. – P. 321–331.
47. Takaoka T. A systematic approach to parallel program verification // Proc. Computing: Australasian Theory Symposium (CATS 96). – Melbourne (Australia), 1996. – P. 48–56.
48. Pnueli A. The temporal logic of programs // Proc. 18th Annual Symposium on the Foundations of Computer Science (Providence). – New York: IEEE Computer Society Press, 1977. – P. 46–57.
49. Chandy K.M., Misra J. Parallel Program Design: A Foundation. – Reading, MA: Addison-Wesley Publishing Company, 1988. – 493 p.
50. Heisel M., Reif W., Stephan W. Implementing Verification Strategies in the KIV-System // Lusk E., Overbeek R. (eds.) Proc. 9th International Conference on Automated Deduction. – Vol. 310 of Lecture Notes in Computer Science. – Berlin: Springer-Verlag, 1988. – P. 216–231.
51. Reif W. The KIV-approach to Software Verification // Broy M., Jähnichen S. (eds.) KORSO: Methods, Languages, and Tools for the Construction of Correct Software. Final Report. – Vol. 1009 of Lecture Notes in Computer Science. – Berlin: Springer-Verlag, 1995. – P. 339–368.
52. Winskel G. The Formal Semantics of Programming Languages: An Introduction. – London: MIT Press Foundations of Computing Series, 1993. – 361 p.
53. Reisig W. The Expressive Power of Abstract-State Machines // Computing and Informatics. – 2003. – Vol. 22, № 3–4. – P. 209–219.
54. Gurevich Y. Sequential Abstract-State Machines Capture Sequential Algorithms // ACM Transactions on Computational Logic. – 2000. – Vol. 1, № 1. – P. 77–111.

55. G. L. Peterson. Myths About the Mutual Exclusion Problem // Information Processing Letters. – 1981. – 12(3). – P. 115–116.
56. Wiley J. Operating Systems Concepts: Seventh Edition. – 2005. – P. 194.
57. Жигалло А.А. Доведення у IPCL коректності алгоритму Пітерсона для взаємного виключення / А.А. Жигалло та ін. // Вісник Київського національного університету імені Тараса Шевченка. Серія: фіз.-мат. науки. – 2015. – вип. 4. – С. 119–124.
58. Zhygallo A.A. Peterson’s Algorithm Total Correctness Proof in IPCL / A.A Zhygallo // Problems of Programming. – 2016. – №2-3. – P. 113–118.