Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Фабунмі Сунмаде Кунле

УДК 004.415,681.3

ФОРМАЛЬНІ МОДЕЛІ КЛІЄНТ-СЕРВЕРНИХ СИСТЕМ В КОМПОЗИЦІЙНИХ МОВАХ ПАРАЛЕЛЬНОГО ПРОГРАМУВАННЯ

01.05.03 – математичне та програмне забезпечення
обчислювальних машин і систем

АВТОРЕФЕРАТ
дисертації на здобуття наукового ступеня
кандидата фізико-математичних наук

Київ-2019

Дисертацією є рукопис.

Робота виконана на кафедрі теорії та технології програмування факультету комп'ютерних наук та кібернетики Київського національного університету імені Тараса Шевченка, МОН України.

Науковий керівник: доктор фізико-математичних наук, професор
Буй Дмитро Борисович.

Офіційні опоненти: доктор фізико-математичних наук, професор
Песчаненко Володимир Сергійович,
Херсонський державний університет,
професор кафедри інформатики, програмної інженерії та економічної кібернетики

кандидат фізико-математичних наук, доцент
Жежерун Олександр Петрович,
Національний університет "Києво-Могилянська академія",
доцент кафедри мультимедійних систем.

Захист відбудеться "30" травня 2019 р. о 14:00 на засіданні спеціалізованої вченої ради Д 26.001.09 Київського національного університету імені Тараса Шевченка за адресою: 03680, м. Київ, проспект академіка Глушкова, 4д, ауд. 01.

З дисертацією можна ознайомитись у Науковій бібліотеці імені М. Максимовича Київського національного університету імені Тараса Шевченка за адресою: 01601, м. Київ, вул. Володимирська, 58, зал № 12.

Автореферат розісланий "26" квітня 2019 р.

Учений секретар
спеціалізованої вченої ради **В.П. Шевченко**

ЗАГАЛЬНА ХАРАКТЕРИСТИКА РОБОТИ

Актуальність теми дослідження. Паралельні середовища сьогодні мають широке застосування, оскільки однопроцесорні системи виявились суттєво обмеженими у швидкодії. Для багатьох сучасних програмних систем при цьому актуальною є потреба доведення їх властивостей, зокрема – коректності функціонування. Одним з найбільших підкласів таких паралельних програмних систем є клієнт-серверні системи. Тому постає питання побудови адекватних моделей паралельних систем, зокрема зі спільною пам'яттю, що дозволяють ефективно виконувати математичну верифікацію властивостей цих систем, а також – зменшують розрив між існуючими теоретичними моделями та реальними паралельними системами шляхом розширення й уточнення перших.

Основною моделлю паралельних систем є виконання з переключенням і взаємодією через спільну пам'ять (англ.: shared memory interleaving concurrency). Це – адекватна модель серверних систем більшості клієнт-серверних комплексів, зокрема, сервісно-орієнтованих, мережевих та інших програмних комплексів.

Для розв'язання вказаних проблем пропонувались різні підходи, зокрема: інсерційне програмування (у роботах акад. Летичевського О.А. та інших), низка моделей та методів, започаткованих Хоаром Т., та розвинутих іншими авторами. Цими питаннями активно займаються підрозділи Інституту кібернетики та Інституту програмних систем НАН України, а також кафедра теорії та технології програмування факультету комп'ютерних наук та кібернетики Київського національного університету імені Тараса Шевченка. На кафедрі розвивається започаткований акад. Редьком В.Н. та проф. Нікітченком М.С. оригінальний композиційний підхід до програмування, що спирається першочергово на семантику мов програмування і має певні переваги над іншими формальними методами.

Це зумовлює актуальність теми дисертаційної роботи, яка присвячена проблемам побудови адекватних моделей програмних систем, зокрема серверних середовищ, з метою доведення коректності функціонування таких систем шляхом специфікації та верифікації. Використання композиційних методів для розв'язку цих проблем дозволяє збагатити засоби специфікації та розширити можливості верифікації на досліджені в роботі класи задач.

Таким чином, актуальність теми дисертаційної роботи зумовлена тим, що клієнт-серверні системи використовуються дуже широко у різних сферах, тому у багатьох випадках вимагається гарантія виконання деяких властивостей систем. В той же час паралелізм, нерозривно пов'язаний з суттю таких систем, додає складності в модель, отже розробка формальних моделей і систем, адекватних клієнт-серверним середовищам, та водночас зручних у використанні – для наступного доведення властивостей таких програм – є важливим питанням комп'ютерних наук.

Зв'язок роботи з науковими програмами, планами, темами. Дисертаційна робота є складовою частиною наукових робіт, проведених на кафедрі теорії та технології програмування факультету комп'ютерних наук та кібернетики Київського національного університету імені Тараса Шевченка при виконанні фундаментальних тем «Формальні специфікації та методи розробки надійних програмних систем» (№ 0111U007052, 2011-2015 рр.), «Розробка логіко-алгоритмічних методів дослідження формальних моделей природних мов» (№ДР0116U004780, 2016-2018 рр.) та «Теорія і методи розробки інтелектуальних інформаційних технологій та систем» (№16КФ015-02, №ДР0116U006378, 2016-2018 рр.).

Мета і задачі дисертаційного дослідження. *Метою* дисертаційної роботи є розробка, аналіз та вдосконалення моделей клієнт-серверних систем в композиційних мовах паралельного програмування.

З огляду на актуальні проблеми у контексті зафіксованої цілі, у даній роботі поставлену загальну мету уточнено до наступних *задач*:

- побудувати розширення композиційних мов специфікацій паралельного програмування для адекватного моделювання клієнт-серверних систем з динамічним породженням програм;
- провести аналіз підходів до породження нових програм у паралельному середовищі та побудувати їх моделі у композиційному програмуванні;
- дослідити метод верифікації властивостей у класі серверних програм на побудованих композиційних моделях;
- виконати апробацію методу на клієнт-серверних системах для верифікації, застосувати запропоновані моделі до відомих систем;
- дослідити складність побудованих моделей.

Об'єктом дослідження є паралельні програми з комунікацією через спільну пам'ять. Об'єкт досліджується переважно з точки зору коректності функціонування.

Предметом дослідження є композиційні моделі, методи та підходи до верифікації паралельних систем.

Наукова новизна одержаних результатів. Запропоновано розширення композиційно-номінативних паралельних мов (Interleaving Parallel Composition Languages, IPCL) та моделі виконання (включаючи модель стану програми), які є більш адекватним поданням для паралельних систем з динамічним породженням програм (що є найбільш поширеним у програмуванні). На підставі уточнення механізму породження паралельних програм побудовано нову композиційно-номінативну модель та обґрунтовано адекватність серверному середовищу класичних клієнт-серверних систем. Побудовано та досліджено клас композиційно-номінативних мов з композиціями породження та приєднання паралельних програм під час виконання, а також подано ряд базових функцій у

цих мовах (атомарні та інші). Для побудованої моделі показано, за яких умов вона буде рівнопотужною з класом мов IPCL. Адаптовано метод верифікації властивостей у класі серверних програм на введених композиційних мовах. Сформульовано та доведено теорему, яка визначає умови, при яких введене розширення моделі еквівалентне існуючій моделі багатоекземплярного виконання у IPCL. Таким чином, наукова новизна полягає у наступному:

- побудовано та досліджено нове розширення композиційних мов специфікацій, що дає можливість адекватно моделювати клієнт-серверні системи з динамічним породженням програм;
- проведено аналіз підходів до породження нових програм у паралельному середовищі, побудовано їх моделі у композиційному підході;
- на підставі уточнення механізму породження паралельних програм побудовано нову композиційно-номінативну модель та обґрунтовано її адекватність серверному середовищу класичних клієнт-серверних систем, а також – прагматичну повноту цієї моделі;
- побудовано та досліджено клас композиційно-номінативних мов з композиціями породження та приєднання паралельних програм під час виконання, а також подано ряд базових функцій у цих мовах (атомарні та інші);
- для побудованої моделі показано, за яких умов вона буде рівнопотужною з класом мов IPCL;
- адаптовано метод верифікації властивостей у класі серверних програм на введених композиційних мовах.

Теоретичне і практичне значення одержаних результатів. Дисертація має теоретико-прикладну спрямованість. Результати роботи можуть використовуватись для специфікації та верифікації паралельних систем з динамічним породженням екземплярів програм, що взаємодіють через спільну пам’ять.

Отримані результати впроваджено у навчальний процес за освітньою програмою «Інформатика» спеціальності 122 «Комп’ютерні науки» на факультеті комп’ютерних наук та кібернетики Київського національного університету імені Тараса Шевченка, а також застосовано для доведення властивостей реальних клієнт-серверних програмних систем.

Особистий внесок здобувача. Всі результати, які складають суть дисертаційної роботи, отримані здобувачем самостійно. З праць, виконаних зі співавторами, на захист виносяться лише результати, отримані особисто здобувачем. У спільно виконаних роботах Д.Б. Бую та Т.В. Панченку належить постановка проблеми, обговорення та інтерпретація результатів. Іншим співавторам – переважно технічні підзадачі з формулювань та доведень окремих тверджень.

Апробація результатів дослідження. Основні положення та висновки дисертаційного дослідження обговорювалися на наукових семінарах кафедри теорії та технології програмування КНУ, семінарі «Програмологія та її застосування».

Результати дисертаційного дослідження оприлюднено у доповідях і повідомленнях на Міжнародних та Всеукраїнських наукових конференціях, семінарах:

- XI International Conference on the Quality of Information and Communications Technology QUATIC 2018 – Coimbra, 4-7.09.2018.
- XIV Міжнародна науково-практична конференція «Теоретичні та прикладні аспекти побудови програмних систем TAAPSD'2017» – Київ, 4-8.12.2017.
- Міжнародна наукова конференція «Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення» Тернопіль, 18.09.2018.
- The First International Conference on Computer Science, Engineering and Education Applications ICCSEEA2018 – Kyiv, 18-20.01.2018.
- XI Міжнародна науково-практична конференція з програмування «УкрПРОГ'2018» – Київ, 22-24.05.2018.
- XII International Conference on Perspective Technologies and Methods in MEMS Design (MEMSTECH). – Lviv, 20-24.04.2016.
- Міжнародний науково-практичний семінар «Комбінаторні конфігурації та їх застосування» – Кропивницький, 15-16.04.2016.
- IV конференція «Комп'ютерне моделювання у наукоємних технологіях» (КМНТ-2016) – Харків, 26-31.05.2016.

Публікації. Результати дисертації опубліковано у 17 наукових працях, з яких 11 – статті [1-11] в наукових журналах і збірниках наукових праць, 6 – у працях та працях і тезах міжнародних наукових конференцій [12-17]; з них 7 статей опубліковано у наукових фахових виданнях, 3 статті опубліковано в іноземних міжнародних наукових журналах, 1 стаття у науковому періодичному виданні іншої держави, 3 публікації у виданнях, що індексується в наукометричних базах Scopus або Web of Science. Основні результати, що виносяться на захист, відображено у роботах [1-6,12-14].

Структура та обсяг дисертації. Дисертаційна робота складається з анотації, переліку умовних позначень, вступу, чотирьох розділів, висновків, списку використаних джерел. Загальний обсяг дисертації становить 161 с, основний текст 127 с., список використаних джерел – 200 найменувань.

ОСНОВНИЙ ЗМІСТ РОБОТИ

У **вступі** обґрунтовується актуальність теми дисертації, визначається об’єкт, мета та завдання дисертаційної роботи, методи дослідження, наукова новизна одержаних результатів та їх теоретико-практичне значення, відображаються апробації та публікації результатів дисертаційного дослідження.

У першому розділі «Розвиток основних понять програмування» розглядаються проблеми розробки ефективних методів побудови програмних систем. Для розвитку адекватного подання програм та судження над ними запропоновано композиційний підхід (композиційне програмування), принципи і ідеї якого висвітлені у першому розділі. Всі побудови беруть початок та ґрунтуються на запропонованому Редьком В.Н. та розвиненому Нікітченком М.С., Буєм Д.Б. та ін. композиційно-номінативному підході до програмування як процесі розробки програм.

Оскільки композиційно-номінативний підхід обрано як методологічну основу подальшого дослідження, його огляду присвячено перший розділ дисертації. Розглянуто просту мову, яку далі буде розширено з метою дисертаційного дослідження на паралелізм.

Підрозділ 1.1 «Проста мова програмування *SIPL* та її формалізація» присвячений огляду універсальної, мови програмування, її складових та принципів. Визначено синтаксис і семантику мови. Розглядається поняття програмної алгебри та семантичного терму, різні типи даних, а саме базові та похідні типи. Наводяться операції на базових типах, на множинах *Int* та *Bool*, операції змішаного типу. В алгебрі мови вирізняють 2 види функцій: *n*-арні функції на базових типах даних та функції над станами змінних (номінативні функції). Введено композиції різних типів.

У **підрозділі 1.2** «Основні принципи композиційно-номінативного підходу» обґрунтовано основні його принципи (включаючи розвиток від абстрактного до конкретного, що використовується при експлікації понять програмування, а також принцип пріоритетності семантики над синтаксисом). У підрозділі наведено обґрунтування академіком Редьком В.Н. важливості композиційності в логіці та програмології. Надалі цей напрям розвинули його учні Нікітченко М.С., Шкільняк С.С., Буй Д.Б., Зубенко В.В., Панченко Т.В.

Формалізовано програмні поняття, використовуючи теоретико-множинний підхід. В основі теоретико-функціонального підходу лежить поняття функції (відображення), яке має розглядатися на різних рівнях абстракції. Це важливо для наступних побудов, оскільки надає гнучкості всьому підходу та дозволяє проводити формальні судження на обраному рівні абстракції (та конкретики), опускаючи не принципові питання на потрібному рівні розгляду на кожному етапі. Вводяться та розглядаються різні класи функцій. Визначення стосуються екстенсіональних аспектів функцій, тобто тих аспектів, які задаються лише через аргументи та значення функцій.

Далі визначаються класи часткових багатозначних функцій, а також програмні системи на їх основі. Для цього задають систему даних, функцій, їх імен, композицій (як функцій вищих порядків) та їх дескрипцій.

Підрозділ 1.3 «Формалізація програмних понять» та **підрозділ 1.4** «Програмні системи» вводять та досліджують основні програмні поняття на трьох

рівнях. «Рівні конкретизації даних» описує три рівні розгляду даних – абстрактний, булевий, номінативний – та, відповідно, три класи систем даних. Номінативні дані є ієрархічно побудованими відображеннями, область визначення яких є множина імен, а область значень – номінативні дані.

«Рівні конкретизації композицій» виділяються три рівні композицій. «Рівні конкретизації дескрипцій» присвячений опису спеціальних класів дескрипцій. Наприклад, якщо композиції є скінченно-арними відображеннями, то імена їх аргументів є натуральними числами, їх впорядкованість можна використати для подання дескрипцій як послідовностей символів. Інтеграція композиційних і дескриптивних систем номінативного рівня за допомогою відображень денотації дозволяє дати цілісний опис різноманітних мов програмування і баз даних.

Такі композиційно-дескриптивні системи номінативного рівня названі композиційно-номінативними системами. Вони дозволяють адекватно описувати семантику різних логічних мов і мов специфікацій прикладних областей.

У розділі 2 «Паралелізм у програмуванні» проаналізовано різні підходи до організації паралельних обчислень та до опису властивостей паралельних програм і суджень над ними. Досліджено різні рівні паралелізму, починаючи від ідей та підходів Ешкрофта, Флойда, Хоара, Харела і Пнуелі та ін.

Оскільки, як зазначено у вступі, клієнт-серверні системи (точніше, їх серверні – основні – частини), які є об’єктом дослідження, є паралельними системами, що виконуються в режимі з переключенням і взаємодіють через спільну пам’ять – саме паралелізму у програмуванні, його різновидам та відомим моделям, присвячено даний розділ дисертації.

У підрозділі 2.1. «Різновиди паралелізму» досліджуються поняття «процес», «паралельні системи», «паралельне програмування», а також виділяються та аналізуються різні види та архітектури паралелізму. З усього різноманіття найбільше акцентується багатопотоковий паралелізм (multithreading, тобто кілька ланцюжків виконання в межах однієї програми) та багатозадачний паралелізм (multitasking, тобто виконання одночасно кількох задач в певному паралельному режимі) як найбільш адекватні моделі для задач дисертаційного дослідження. З двох парадигм паралелізму (двох підходів до міжпроцесових комунікацій) – взаємодії через спільну пам’ять (shared memory, shared variables) та взаємодії шляхом передачі повідомлень (message passing, communication) – перша є більш адекватною задачам даної роботи, в той час як друга є більш дослідженою іншими авторами.

У підрозділі 2.2. «Паралелізм шляхом обміну повідомленнями та паралелізм через спільну пам'ять» розглядається різниця між підходами. У другому випадку взаємодія між процесами часто є опосередкованою і проявляється автоматично та (іноді) непередбачувано через доступ до спільної пам’яті.

У підрозділі порівняно дослідження щодо управляючих просторів в асинхронних паралельних обчисленнях Анісімова А.В., розпочаті разом з Глушковим В.М., технологія програмування ПАРКС (паралельні асинхронні рекурсивно-керовані системи), а також низку публікацій та монографій академіка Летичевського О.А., академіка Андона П.І. та професора Лаврищевої К.М. щодо розподілених систем, професорів Кривого С.Л. та Чеботарева А.М. щодо методів аналізу властивостей реактивних систем. Всі ці розробки в першу чергу

орієнтовані на розподілені системи та протоколи, і, відповідно, вони орієнтовані на взаємодію процесів (агентів) через механізм обміну повідомленнями.

У підрозділі 2.3. «Підходи до паралелізму через спільну пам'ять в режимі почергового виконання» згадуються напрацювання Хоара (Hoare C.A.R.), Овіцкі (Owicki) та Гріса (Gries), які зробили спробу побудувати систему висновків щодо паралельних програм шляхом узагальнення методу Хоара. Вони розглядали структуровані програми з паралелізмом (режим почергового виконання), що виражається оператором **cobegin** $S_1 \parallel S_2 \parallel \ldots \parallel S_n$ **coend**. Запропоноване ними додаткове правило дозволяє, згідно традицій методу Хоара, зводити доведення властивостей всієї програми до доведення подібних властивостей для окремих операторів програми.

Далі у підрозділі детально розглядається метод Овіцкі-Гріса. Він дозволяє використовувати додаткові змінні для фіксації інформації про взаємодію. Такі вихідні положення привели до плутанини та складнощів при застосуванні цього методу до реальних систем. Метод Овіцкі-Гріса зводить доведення до верифікаційних умов для кожного оператора програми. Проте для програми з n операторами утворюється $O(n^2)$ умов верифікації замість $O(n)$ умов у методі Флойда. Як розвиток методу Джонс (Jones) запропонував включити цю інформацію в специфікацію шляхом додавання двох предикатів *rely* та *guar*, які описують зміну стану середовищем (тобто всіма іншими процесами) та самим процесом відповідно. *rely* описує, на що може покладатись процес (rely on, твердження про середовище, факти), а *guar* (від guarantee) описує те, що він гарантує. Введений Джонсом формалізм потрапляє до категорії rely–guarantee або assumption–commitment методів. Цю спробу зроблено, аби перетворити метод Овіцкі-Гріса в композиційний.

У розділі розглядається й інша гілка розвитку цього підходу – мова програмування Unity Ченді (Chandy) та Місри (Misra), які розробили її без структури управління. Програми в цій мові – це нескінчений (поки щось може виконатись) цикл оператору вибору. Для здійснення суджень про Unity-програми, Ченді та Місра розробили логіку Unity. Вони запропонували правила виводу для формалізації доведень переважно інваріантів та властивостей типу «веде-до».

Недолік мови Unity – недостатня виразність її логіки, що є наслідком простоти конструкцій мови. Це знову зумовлює необхідність залучення додаткових змінних, присутніх лише в специфікації та відсутніх в реалізації, які формально не відрізняються від звичайних змінних, а це веде до пов’язаних непорозумінь.

Підрозділ 2.4. «Методи суджень про паралельні програми та їх класифікація» присвячений опису різних підходів щодо паралельних систем. Лемпорт (Lamport) відзначає два загальні шляхи дослідження виконання програми: як послідовність станів (state-based), або як послідовність подій (event-based) чи дій (action-based).

В той час як методи, що базуються на станах, використовують підхід, який базується на твердженнях, судженнях (виводі) та понятті інваріанта (assertional approach with concept of invariance) до верифікації, в алгебраїчному підході верифікація базується на застосуванні алгебраїчних перетворень (algebraic

transformations), а в функціональному підході – використовуються правила застосування функцій (functional application).

В аксіоматичному підході, який теж базується на станах, модель є абстрактною і задає перелік властивостей системи. Лемпорт зазначає, що коли виписані всі властивості, тобто описано, що система повинна і не повинна робити, то ми отримуємо абстрактну специфікацію. За нею важко уявити цілковиту картину, складно визначити, які додаткові властивості випливають та не випливають з цього переліку. В результаті стає неможливим визначити, чи повно описує систему специфікація. Особливо ці зауваження стосуються більш-менш складних систем. Тому Лемпорт робить висновок, що аксіоматичні методи в чистому вигляді не працюють.

Лемпорт зауважує, що ситуація потенційно значно покращується, якщо специфікаціями є програми на деякій абстрактній мові. Тоді з'являється можливість формально вводити такі поняття як початковий стан, функція переходу (next-state relation) та доводити звідність (implementation relation). Водночас Лемпорт зауважує, що зазвичай немає точного визначення самої мови моделювання, немає чітко визначеної формальної семантики і правил виводу. Це, зокрема, стосується мови UML. Причиною є складність такої формалізації, тому багато що робиться «ad hoc». Як приклад Лемпорт наводить мову Unity.

У розділі виділяються дві гілки методів перевірки моделей: перевірка темпоральних моделей (метод базується на темпоральній логіці) та автоматне моделювання (включає порівняння поведінки автомата-специфікації та автомата-моделі системи). На відміну від підходу theorem proving (див. далі), перевірка моделі є автоматичною процедурою, вона використовується для перевірки часткової коректності та може продукувати контрприклади, вказуючи на помилки у розробці. Основний недолік методу перевірки моделі – проблема «вибуху» кількості станів, тобто експоненційного росту їх кількості – state explosion problem. Розроблено низку підходів для боротьби з цією проблемою. Зокрема, це символьна перевірка моделей (symbolic model checking), семантична мінімізація та інші. Проте проблему «вибуху» повністю не вирішено. Методи символьної перевірки моделей мають лише евристичну перевагу, але не зменшують загальну складність перевірки моделей.

У розділі надається визначення терміну «Доведення теорем» та методи доведення: керовані користувачем (user-guided automatic deduction tools: ACL2, LP та інші), перевірники доведень, або прувери (proof checkers: HOL, LCF, Nurpl, інші) та комбіновані системи (combination provers: PVS, STeP, Analytica тощо). Такі системи зазвичай підтримують механізм прийняття рішень (decision procedures). Цей підхід може оперувати нескінченною кількістю станів. Доведення теорем спирається, як правило, на техніки типу структурної індукції (structural induction) для здійснення доведень над нескінченними доменами (prove over infinite domains).

Окрім аксіоматичних та операційних методів, Лемпорт окремо виділяє ще один підхід, базований на апараті темпоральної логіки. Цей підхід є розвитком ідей Ешкрофта щодо інваріанта. Зазначений підхід пропонує розглядати специфікації як абстрактні програми у вигляді математичних формул.

Таким чином, розглянувши різноманіття напрацювань та їх недоліки, підведено підґрунтя під розширення мови *SIPL* (Simple Imperative Programming Language) конструкціями для паралелізму, що слугуватимуть адекватною моделлю клієнт-серверних систем – тобто паралельних програм, що взаємодіють через спільну пам'ять. Це дозволить використати потенціал та напрацювання у композиційно-номінативному підході для побудови адекватних формальних моделей і подальшого судження над паралельними програмами, які взаємодіють через спільну пам'ять і доведення їх властивостей. Зокрема, для доведення коректності таких програм відносно специфікацій.

Третій розділ «Доведення властивостей програм в композиційно-номінативних мовах IPCL» присвячений опису методу доведення властивостей програм в композиційних мовах *IPCL*. Введено розширення *SIPL* на інструкції паралельного виконання та надано методику доведення коректності паралельних програм. Визначена оцінка складності доведення відносно кількості операторів програми. Встановлені основні переваги над іншими підходами та наведено приклади використання. Мова *IPCL* вважається актуальною моделлю клієнт-серверних систем, як зазначено в ряді публікації автора.

В підрозділі 3.1 «Актуальність використання паралелізму зі спільною пам'яттю» розглянуто актуальність питання розробки формального методу для доведення властивостей систем, що взаємодіють через спільну пам'ять. Розглядаються підходи до вирішення цієї проблеми.

Як зазначалось вище, аксіоматичні методи є складними у застосуванні на практиці щодо паралельних систем. Тому, для таких програм, доцільно надати перевагу методам, що базуються на моделюванні та використовують принцип інваріанта (постійної властивості) у своїй основі.

Незважаючи на те, що багатопотоковість (multithreading) є складною та схильною до помилок технікою програмування, яка рекомендується до використання лише для найпростіших програм (за Хоаром), нас цікавить саме ця техніка взаємодії у контексті дослідження. Так, Хоар зазначив, що паралелізм може бути дозволений лише на найбільш зовнішньому (найбільш глобальному, найвищому) рівні (завдання, програми), а його використання на нижчих (вкладених) рівнях треба уникати.

На користь застосування паралелізму зі спільною пам'яттю свідчить сучасна ситуація з *hardware*, зокрема, архітектура *SMD* багатоядерних процесорів.

В підрозділі 3.2 «Мова IPCL. Синтаксис» розглянуто синтаксис мови IPCL. Він задається наступним чином:

$P ::= \bar{x} := \bar{e} \mid P_1;P_2 \mid$ **if** b **then** P_1 **else** $P_2 \mid$ **while** b **do** $P \mid P_1 \parallel P_2$.

Тут оператор присвоювання є векторним та атомарним, композиції послідовного виконання, розгалуження та циклування розуміються стандартним чином, а композиція || – позначає паралельне виконання.

В підрозділі 3.3 «Композиційна семантика мов класу IPCL» розглянуто композиційну семантику мов класу IPCL.

Мови класу *IPCL* ґрунтуються на композиційно-номінативних системах вигляду <*NDS, F,* {1,2}, *C, arity*>, де

– *NDS* – система іменних або номінативних даних (надалі – просто дані),

– $F = Oper \cup Pred$ – набір функцій для перетворення даних,

– C – композиції над функціями з F.

Маємо $Oper = D \times D \to D \times D$ та $Pred = D \times D \to \{True, False\}$, де перше входження D в декартів добуток – «глобальні дані», а друге – «локальні» для поточного процесу. Тут $D = ND(V,W)$ – множина номінативних даних.

Функції, що повертають значення з множини $\{True, False\}$, тобто предикати, не змінюють поточний стан даних $D \times D$ – вони використовуються в якості умови в операторах розгалуження та циклування.

Глінн Вінскел вказує, що неможливо змоделювати паралельне виконання програми, обмежившись зв'язками між конфігураціями команд та заключними станами. Натомість необхідно використовувати зв'язки між окремими атомарними (неподільними) кроками при виконанні і таким чином дозволяти одній команді впливати на стан іншої команди, з якою вона виконується в паралель. Саме такий підхід і застосовано у даній роботі. Семантика систем з паралельним виконанням в режимі чергування зазвичай визначається саме в синтактико-семантичних (операційних) традиціях, адже цей підхід є більш природним для подібних систем.

Функції з множини F є атомарними транзакціями, неподільними в сенсі паралелізму – їх виконання не може бути перервано. Зокрема, умови у відповідних композиціях є теж атомарними. Конкретна ж мова класу *IPCL* утворюється шляхом фіксації множини F.

Твердження 3.1. Семантика *IPCL* описана повно. Для доведення використано індукцію по структурі програми.

Показано, що семантика мови *IPCL* є композиційною, коректною та повною – семантично та прагматично.

В підрозділі 3.4 «Два види властивостей програм *IPCL*» розглянуто два види властивостей програм *IPCL* – узагальнена часткова коректність програми та властивість типу інваріанту (одного для всієї програми).

Обидва варіанти властивостей є варіаціями поняття властивості «безпеки» (safety property). Перший варіант відображає традиційні погляди (зокрема, Хоара) на часткову коректність послідовних програм, другий варіант лежить в руслі переходу до інваріанту замість перед- та пост-умов і відображає погляди Ешкрофта-Лемпорта на коректність паралельних програм.

В підрозділі 3.5 «Методика доведення властивостей програм *IPCL*» наведено спосіб доведення властивостей програм *IPCL*, враховуючи, що кожна підпрограма може взаємодіяти зі спільними глобальними та своїми локальними даними. Розглянуто транзиційну систему та побудову відповідної моделі виконання програми $P \in SeqILProgs$. Розглядаються програми загального вигляду: $P = A^n \parallel B^m \parallel \ldots \parallel C^k \in SeqILProgs$.

Через розмітку цих підпрограм будується спільна транзиційна система для довільної фіксованої кількості екземплярів програм $A, B, \ldots, C$ з відповідними множинами міток $A_{marks}, B_{marks}, \ldots, C_{marks}$. Станами системи для програми P будуть $S \in A_{marks}^{\,n} \times B_{marks}^{\,m} \times \ldots \times C_{marks}^{\,k} \times D \times D^{n+m+\ldots+k}$. Тут перше входження D в декартів добуток – «глобальні дані», а друге – «локальні» для всіх окремих послідовних підпрограм (A^n, B^m, …, C^k) з P, записані в тій же послідовності, що і самі підпрограми в термі програми P. Перша ж частина декартового добутку (до

даних) – це позиції виконання кожної підпрограми з *P*, мітки функцій, які будуть обчислені відповідними підпрограмами, коли дійде черга до їх виконання, тобто відбудеться «переключення на них», на найближчому кроці (класичні правила interleaving). Всі такі стани являють собою множину станів (*States*).

Початкові стани $StartStates \subseteq States$ та заключні *StopStates* –для всіх елементів *s* яких характерно, що перші їх компоненти будуть мати значення заключних (останніх в записі терму програми) міток відповідних процесів, а також вимагається їх досяжність. Останнє означає, що для кожного стану $s \in StopStates$ існує набір станів $s_1, s_2, \ldots, s_l$ такий, що $s_1 \in StartStates$ & $s_l = s$ & $(\forall i \in \{1, 2, \ldots, l-1\} \bullet (s_i, s_{i+1}) \in Step)$, де функція *Step* – функція кроку виконання програми *P* (всі допустимі переходи у транзиційній системі, за один крок під час виконання програми *P*, тобто атомарні дії).

З виконанням кожного кроку програми *P*, тобто обчисленням деякої функції, окрім можливої зміни даних, пов'язана зміна значення однієї з перших компонент стану, а саме компоненти, що вказувала на мітку щойно обчисленої функції. Її значення стає рівним мітці, куди програма потрапляє після обчислення функції з поточного кроку. Значення нової мітки визначається за семантикою композиційного контексту даної функції. Функція *Step* є багатозначною недетермінованою частковою функцією $Step \subseteq States \times States$ і являє собою ініціалізовану нескінченну (за кількістю станів) нерозмічену транзиційну систему (Abstract State Machine). Зрозуміло, що для кожної програми з *SeqILProgs* можна побудувати відповідну транзиційну систему.

Наведено алгоритм розстановки міток та визначення функції *Step*. Тепер можна промоделювати виконання всієї програми *P* як почергове виконання деякої з поточних операцій (обчислення значень функцій), зокрема, перевірку умови і перехід у відповідності з результатом до блоку «then» або до блоку «else», та відповідну зміну стану. Апарат моделювання є більш потужним, ніж запропонована семантика *IPCL*. Він виявляється достатнім для моделювання більш складних конструкцій з динамічним породженням у розділі 4. Також розглянуто спрощену модель (за відсутності локальних даних).

В підрозділі 3.6 «Доведення властивостей програм *IPCL»* розглянуто підхід до доведення властивостей в *IPCL*. Ми дозволяємо явно використовувати стан управління у формулюванні предикатів *PreCond, PostCond* та *Inv* – для формулювання відповідних властивостей. Показано, що властивості обох типів є взаємо-звідними, або рівнопотужними:

Твердження 3.2. Властивість 1 типу виконується для деяких *PreCond*(*S*), *PostCond*(*S*) та програми *P* тоді і тільки тоді, коли виконується властивість 2 типу для цієї програми *P* і деякого інваріанта *Inv*(*S*).

Теорема 3.1. Нехай програма *P* (функція переходу *Step**) переводить початковий стан *S* ($S \in StartStates$) в кінцевий стан *S*' ($S' \in StopStates$).

Тоді для будь-якої пари таких станів *S* та *S*' з *PreCond*(*S*) випливає *PostCond*(*S*') тоді і тільки тоді, коли існує інваріант *Inv*(*s*) ($s \in States$), який є логічним наслідком *PreCond*(*s*) для кожного початкового стану ($\forall s \in StartStates \bullet (PreCond(s) \rightarrow Inv(s))$). З нього випливає *PostCond*(*s*) в кожному заключному стані ($\forall s \in StopStates \bullet (Inv(s) \rightarrow PostCond(s))$) і він

зберігає істинність для кожного переходу з *Step* ($\forall(s, s')\in Step \bullet (Inv(s) \rightarrow Inv(s'))$), або:

$\forall Step, States, StartStates, StopStates \bullet \forall PreCond \bullet \forall PostCond \bullet (\forall S\in StartStates \bullet \forall S'\in StopStates \bullet (PreCond(S) \,\&\, (S, S')\in Step^* \rightarrow PostCond(S')) \Leftrightarrow \exists Inv \bullet InvCond(Inv, PreCond, PostCond))$.

У роботі надано схему доведення властивостей в *IPCL*: за програмою в мові *IPCL* (згідно наведеного алгоритму) будуємо функцію *Step* усіх можливих переходів (та розставляємо мітки); фіксуємо множини початкових та кінцевих станів (згідно наведених визначень) – *StartStates* та *StopStates*; формулюємо перед- та постумови та знаходимо за ними відповідний інваріант, або ж одразу формулюємо властивість типу інваріанту; маючи $Inv(S)$, $S\in States$, робимо доведення-перевірку, що інваріант-предикат зберігає свою істинність над усіма можливими переходами *Step*.

Далі наведено приклади застосування методу доведення властивостей в *IPCL* – починаючи від простих (паралельне додавання до спільної комірки) до доведення властивостей частин функціональності комерційних систем. Надається порівняльна характеристика щодо інших методів, зокрема по складності самих доведень відносно розміру програми.

Зроблено висновки щодо методу доведення властивостей в *IPCL*, його застосовності, відмінностей та переваг. Суттєвим недоліком методу є необхідність формулювання інваріанту для програми, адже він відіграє в методі одну з ключових ролей (та і в судженні над паралельними програмами в цілому, згідно з Лемпортом).

При цьому, метод більш суттєво зв’язаний з текстом програми на мові *IPCL*, яка схожа на звичайні мови імперативного програмування (на яких, власне, і пишеться переважна більшість реального програмного забезпечення), на відміну від багатьох інших, які стартують судження від логічних формул, створюючи тим самим розрив між світами логіки та програмування. Клас програм, які розглядаються в рамках наведеного методу, в явному вигляді не розглядається в жодному іншому підході, а в більшості взагалі не матиме адекватного подання.

Четвертий розділ «Багатопотокові моделі композиційної мови *IPCL*» присвячений введенню моделі паралелізму в *IPCL* з динамічним породженням екземплярів (паралельних) підпрограм. В розділі описано розширений динамічний стан та модель породження екземплярів у мові *IPCL*. Введено відповідні базові функції *start* та *join*, задано їх семантику. Описано семантику функцій *fork*, *lock*, *unlock*, важливих для практичного використання *multithreading* паралелізму в транзиційній моделі *IPCL*. Особливе місце в розділі посідає визначення семантики атомарних операцій *test_and_set*, *swap* та узагальненої операції *atomic*. Зазначені операції є важливими як для організації взаємодії паралельних процесів і підпрограм, так і для розширення мов *IPCL* в цілому. Наведено теорему про еквівалентність (рівнопотужність) двох підходів до побудови програм і, відповідно, двох семантик – розширеної (розділ 4) та базової (розділ 3). Показано, що у випадках програм, що завершуються, можлива взаємна виразність.

В підрозділі 4.1 «Системи паралельного виконання програм в *IPCL*» розглянуто дві наведені моделі паралелізму у *IPCL*. В підрозділі стверджується

функціональна еквівалентність двох систем. Програма в даному випадку розглядається як функція над даними. Тобто, для довільної програми в одній з систем паралелізму можна побудувати відповідну їй програму в іншій системі, яка повертає той самий результат (тобто є функціонально еквівалентною). У контексті взаємної виразності ми розглядаємо лише продуктивні програми, тобто такі, що завершуються і повертають дане, яке містить результат їх роботи (обчислень). Програми, що «зациклюються», тобто не завершуються, не є предметом дослідження даної роботи. Із функціонального погляду такі програми не повертають ніякого результату над вхідними даними. Іншими словами, ми говоримо про тотальні програми.

У **підрозділі 4.2** «Модель динамічного стану транзиційної моделі із породженням та приєднанням паралельних програм». Введено модель динамічного стану транзиційної моделі із породженням та приєднанням паралельних програм. Модифікований стан транзиційної моделі *IPCL* подано у вигляді:

$$S = \left(\left(\begin{pmatrix} m_{11} \\ m_{12} \\ \vdots \\ m_{1n_1} \end{pmatrix}, \begin{pmatrix} m_{21} \\ m_{22} \\ \vdots \\ m_{2n_2} \end{pmatrix}, \dots, \begin{pmatrix} m_{k1} \\ m_{k2} \\ \vdots \\ m_{kn_k} \end{pmatrix} \right), \left(d, \begin{pmatrix} d_{11} \\ d_{12} \\ \vdots \\ d_{1n_1} \end{pmatrix}, \begin{pmatrix} d_{21} \\ d_{22} \\ \vdots \\ d_{2n_2} \end{pmatrix}, \dots, \begin{pmatrix} d_{k1} \\ d_{k2} \\ \vdots \\ d_{kn_k} \end{pmatrix} \right) \right)$$

Структура стану є схожою на попередню, але тепер дозволяє динамічне додавання або видалення компонентів без проблем для арності (вектор векторів). Таким чином, в стані *S* за один крок може змінитись три компоненти (мітка m_{ij}, глобальне *d* або локальне дане d_{ij} для програми P_{ij}). Сукупну множину всіх станів можна задати для певної фіксованої кількості різновидів підпрограм k таким чином:

$$States = \bigcup_{n_1, n_2, \dots, n_k \in N} \{ s \in \left((M_1^{n_1}, M_2^{n_2}, \dots, M_k^{n_k}), (D, D^{n_1}, D^{n_2}, \dots, D^{n_k}) \right) \}$$

Інакше кажучи, множину всіх станів *States* задаємо як об'єднання множин станів для всіх можливих кількостей екземплярів паралельних підпрограм $P_i, i \in \{1, \dots, k\}$, тобто для всіх комбінацій натуральних степенів відповідних множин міток M_i та множини даних *D*.

У **підрозділі 4.3** «Семантика операцій *test_and_set* та *swap* в транзиційній моделі виконання програм *IPCL*» визначено семантику операцій *start*() та *join*() створення нового екземпляру паралельної підпрограми та приєднання створеної раніше (тобто, очікування її завершення), а також інших базових функцій розпаралелення та атомарної синхронізації – *fork*(), *lock*(), *unlock*(). Далі наведено семантику операцій *test_and_set*() та *swap*() в транзиційній моделі виконання програм *IPCL*. Це дає можливість виражати у IPCL ширший клас паралельних програм і їх способів взаємодії та синхронізації.

Також визначено семантику атомарного обчислення *atomic*(*P*) для довільної програми *P* – без переключення від старту до завершення, тобто без передачі управління іншим паралельним програмам.

Подібним чином можна задати семантику функцій роботи з семафорами (зокрема, функції Дейкстри *P*(*semaphore*) і *V*(*semaphore*), або *wait*(*semaphore*) і *signal*(*semaphore*) відповідно) та інших, що є основою атомарного виконання і механізмів синхронізації та управління доступом в операційних системах, зокрема, *enter_critical_section*(*section_id*) та *exit_critical_section*(*section_id*), моніторах тощо.

Всі побудовані конструкції є важливими для збагачення *IPCL* з метою адекватного покриття сучасних засобів конструювання паралельних клієнт-серверних програм.

В підрозділі 4.4 «Еквівалентність двох систем паралельного виконання програм» наведено відповідну теорему.

Розглядаються лише тотальні програми, адже функціонально інші програми не є коректними, оскільки вони не повертають ніякого результату над вхідними даними та, звідси, не являють користі у практичному сенсі (прикладному програмуванні).

Теорема 4.1. Дві семантичні моделі виконання – *IPCL* $P_1^{n_1} || P_2^{n_2} || \dots || P_k^{n_k}$ (з фіксованими степенями паралельних програм) та модель з динамічним породженням паралельних підпрограм *start*() / *join*() (специфікована в даному розділі, див. вище) еквівалентні за обчислювальною потужністю.

ВИСНОВКИ

В дисертаційній роботі проведено ретельний порівняльний аналіз існуючих формальних методів, застосовних до проблеми, а також запропоновано більш адекватну модель паралельного виконання із породженням екземплярів програм. Проведено дослідження запропонованого розширення та надано його характеристику.

Головний результат дисертаційної роботи – розробка композиційних моделей, розширення мови специфікації та методів верифікації програмних систем, що розв’язують практично важливі задачі специфікації клієнт-серверних систем та верифікації часткової коректності даного класу програмного забезпечення у галузі програмного забезпечення обчислювальних машин і систем (або ж – комп’ютерних наук). Розроблені моделі та розширення методів мають суттєве значення для підвищення якості та надійності програмних систем.

В результаті проведеної роботи вирішено такі наукові задачі:

1. Побудовано та досліджено нове розширення композиційних мов специфікацій, що дає можливість адекватно моделювати клієнт-серверні системи з динамічним породженням програм.
2. Проведено аналіз двох підходів до породження нових програм у паралельному середовищі, сконструйовано їх моделі у композиційному підході. Показано переваги обох підходів.

3. Шляхом уточнення механізму породження паралельних програм створено нову композиційно-номінативну модель та обґрунтовано адекватність серверному середовищу класичних клієнт-серверних систем. Це дає можливість не тільки більш точно відображати поведінку клієнт-серверних систем, а й підвищує виразність мови для моделювання поведінки таких паралельних систем.
4. Визначено та досліджено клас композиційно-номінативних мов з композиціями породження та приєднання паралельних програм під час виконання, а також визначено базові функції в рамках цих мов, що дозволило розширити засоби подання паралельної взаємодії програм.
5. Для побудованої моделі доведено, за яких умов вона буде рівнопотужною з класом паралельних композиційних інтерлівінгових мов (IPCL), що дозволяє спростити судження над програмами у таких випадках.
6. Адаптовано метод верифікації властивостей для класу серверних програм на введених композиційних мовах, що дозволяє ефективно проводити доведення властивостей у новому класі мов.
7. Для обґрунтування ефективності запропонованого методу його апробовано на актуальних клієнт-серверних системах для їх верифікації.

СПИСОК ПРАЦЬ, ОПУБЛІКОВАНИХ ЗА ТЕМОЮ ДИСЕРТАЦІЇ

- **Статті у фахових виданнях України та наукових періодичних виданнях інших держав:**

1. Фабунмі С.К. Семантика деяких атомарних операторів у IPCL / Sunmade Fabunmi // Вісник Київського національного університету імені Тараса Шевченка. Серія фізико-математичні науки. – 2017. – №4. – С. 59-62.
2. Fabunmi Sunmade. On Two Representations of Concurrent Programs / Sunmade Fabunmi, T. Panchenko // International Journal "Information Models and Analyses". – Sofia, Bulgaria, ITHEA. – 2017. – Vol. 6, No. 2. – pp. 192-199
3. Fabunmi Sunmade. IPCL Extension For Spawning Concurrency / Sunmade Fabunmi // The Scientific Heritage. – Budapest, Hungary. – 2018. – No. 26, Vol. 2. – pp. 51-53
4. Фабунмі С.К. Еквівалентність двох систем паралельного виконання / Sunmade Fabunmi, Т.В. Панченко // Проблеми програмування. – 2018. – №2-3. – С. 93-98.
5. Fabunmi Sunmade. On Formalization of Semantics of Real-time and Cyber-Physical Systems / Sunmade Fabunmi, Ie. Ivanov, M. Nikitchenko, T. Panchenko // Advances in Computer Science for Engineering and Education (Advances in Intelligent Systems Series, Z.B. Hu, S. Petoukhov, I. Dychka, M. He Eds.). – 2018. – Issue 754. – pp. 213-223

6. Фабунмі С.К. Розширений динамічний стан та модель породження екземплярів у IPCL / Sunmade Fabunmi, Є.В. Іванов, Т.В. Панченко, А.В. Скідоненко, Є.О. Трофименко // Вісник Київського національного університету імені Тараса Шевченка. Серія фізико-математичні науки. – 2017. – №4. – С. 127-130.
7. Фабунмі С.К. Логики частичных предикатов, индуцированные трехзначными логиками Клини / Д. Б. Буй, Е. В. Шишацкая, К. Д. Мухаммед, С. К. Фабунмі // Штучний інтелект. – 2015. – №3-4. – С. 84-88.
8. Фабунмі С.К. Відношення конфінальності, передпорядки та порядки, семантика фрази ORDER BY запитів SQL-подібних мов [Електронний ресурс] / Д. Б. Буй, Н. Д. Кахута, О. В. Шишацька, Karam Jasim Mohammed, Sunmade Fabunmi // Вісник Київського національного університету імені Тараса Шевченка. Серія: Фізико-математичні науки. – 2015. – Вип. 4. – С. 88-95. – Режим доступу: http://nbuv.gov.ua/UJRN/VKNU_fiz_mat_2015_4_16
9. Фабунми С.К. Математические основания множественного наследования: рефлексивно-транзитивное замыкание. / Д.Б. Буй, Е.В. Шишацкая, Ф. Санмейд // Вісник Харківського національного університету ім. В.Н. Каразіна. – 2016. – №28. – С. 19-33 http://periodicals.karazin.ua/mia/article/viewFile/6549/6058
10. Фабунмі С.К. Рефлексивно-транзитивные замыкания бинарных отношений. / Д.Б. Буй, Е.В. Шишацкая, Ф. Санмейд // Электротехнические и компьютерные системы. – 2016. – № 22 (98). – С. 272-276. http://www.etks.opu.ua/?fetch=articles&with=info&id=806
11. Fabumni S. Mathematical foundations of linearization algorithms: reflexive-transitive closures of binary relations. / Дмитрий Борисович Буй, Елена В. Шишацкая, Fabumni Sunmade, Mohammed Karam Jasim // Вісник Харківського національного університету імені В. Н. Каразіна. Серія: Математичне моделювання. Інформаційні технології. Автоматизовані системи управління. – 2016. – Том 29 (2016). – С. 19-33. (http://periodicals.karazin.ua/mia/issue/view/473)

- **Тези доповідей на міжнародних конференціях:**

12. Фабунмі С.К. Модель паралелізму в IPCL з породженням екземплярів / Sunmade Fabunmi, Є. Іванов, Т. Панченко // Міжнародна конференція "Теоретичні та прикладні аспекти побудови програмних систем TAAPSD'2017" (Україна, Київ, 4-8.12.2017). - 2017. - С. 131-133
13. Fabunmi Sunmade. Quality of Concurrent Shared Memory Programs [Electronic publication] / Sunmade Fabunmi, T. Panchenko // International Conference on the Quality of Information and Communications Technology QUATIC 2018 (Portugal, Coimbra, 4-7.09.2018). - 2018. – С. 299-300

14. Фабунмі С.К. Композиційні моделі клієнт-серверних систем / Sunmade Fabunmi // Міжнародна наукова конференція «Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення», вип. 31, (Україна, Тернопіль, 18.09.2018). - 2018. - С. 57-60
15. Fabunmi S. Mathematical Foundations of Multiple Inheritance: Reflexive-transitive Closure of the Binary Relations / Dmytro Buy, Olena Shyshatska, Sunmade Fabunmi, Karam Mohammed // Perspective Technologies and Methods in MEMS Design (MEMSTECH), 2016 XII International Conference on Perspective Technologies and Methods in MEMS Design (MEMSTECH). – 2016. – С.192-195. (http://ieeexplore.ieee.org/document/7507540/)
16. Фабунми С.К. Математические основания множественного наследования: рефлексивно-транзитивное замыкание / Д.Б. Буй, Е.В. Шишацкая, Ф. Санмейд // Міжнародний науково-практичний семінар – "Комбінаторні конфігурації та їх застосування" – 2016. http://it-kntu.kr.ua/2016/04/18/results-cca-2016/#more-3782
17. Фабунми С.К. Математические основы множественного наследования в ООП / Буй Д.Б., Шишацкая Е.В., Fabunmi S. // Четвёртая конференция «КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ В НАУКОЕМКИХ ТЕХНОЛОГИЯХ» (КМНТ-2016). – Харьков, 26-31 мая 2016 г (http://www.dsmmph.org.ua/kmnt.html)

АНОТАЦІЯ

Фабунмі С.К. Формальні моделі клієнт-серверних систем у композиційних мовах паралельного програмування. – Рукопис.

Дисертація на здобуття наукового ступеня кандидата фізико-математичних наук за спеціальністю 01.05.03 – математичне та програмне забезпечення обчислювальних машин і систем. – Київський національний університет імені Тараса Шевченка, – Київ, 2019.

Основна модель паралельних систем, що досліджуються, є виконання з переключенням і взаємодією через спільну пам’ять (shared memory interleaving concurrency). Запропоноване розширення мови IPCL та моделі виконання (включаючи модель стану програми) є більш адекватним поданням для паралельних систем з динамічним породженням програм (що є найбільш поширеним у програмуванні). На підставі уточнення механізму породження паралельних програм побудовано нову композиційно-номінативну модель та обґрунтовано адекватність серверному середовищу класичних клієнт-серверних систем, а також – прагматичну повноту цієї моделі. Побудовано та досліджено клас композиційно-номінативних мов з композиціями породження та приєднання паралельних програм під час виконання, а також подано ряд базових функцій у цих мовах (атомарні та інші). Для побудованої моделі показано, за яких умов вона

буде рівнопотужною з класом мов IPCL. Адаптовано метод верифікації властивостей у класі серверних програм на введених композиційних мовах. Сформульовано та доведено теорему, яка визначає умови, за яких введене розширення моделі еквівалентне існуючій моделі багатоекземплярного виконання у IPCL. Отримані результати впроваджено у навчальний процес, а також застосовано для доведення властивостей клієнт-серверних програмних систем.

Ключові слова: композиційне програмування, композиційно-номінативні мови, доведення властивостей програм, формальні методи верифікації, клієнт-серверні системи, паралельні програми, паралелізм зі спільною пам'яттю.

АННОТАЦИЯ

Фабунми С.К. Формальные методы клиент-серверных систем в композиционных языках параллельного программирования. – Рукопись.

Диссертация на соискание ученой степени кандидата физико-математических наук по специальности 01.05.03 – математическое и программное обеспечение вычислительных машин и систем. – Киевский национальный университет имени Тараса Шевченко, – Киев, 2019.

Основной моделью параллельных систем, исследуемой в диссертации, является выполнение с переключением и взаимодействием через общую память (shared memory interleaving concurrency). Предложенное расширение языка IPCL и модели выполнения (включая модель состояния программы) является более адекватным представлением для параллельных систем с динамическим порождением программ (что является наиболее распространенным в программировании). На основании уточнения механизма порождения параллельных программ построена новая композиционно-номинативную модель и обоснованно адекватность серверной среде классических клиент-серверных систем, а также – прагматическую полноту этой модели. Построены и исследованы класс композиционно-номинативных языков с композициями порождения (start) и присоединения (join) параллельных программ во время выполнения, а также представлен ряд базовых функций в этих языках (атомарные и другие). Для построенной модели показано, при каких условиях она будет эквивалентна модели многоэкземплярного выполнения в языках IPCL. Адаптирован метод верификации свойств в классе серверных программ на введенных композиционных языках. Полученные результаты внедрены в учебный процесс, а также применены для доказательства свойств клиент-серверных программных систем.

Ключевые слова: композиционное программирование, композиционно-номинативные языки, доказательство свойств программ, формальные методы верификации, клиент-серверные системы, параллельные программы, параллелизм с общей памятью.

ANNOTATION

Fabunmi S.K. Formal Methods for Client-Server Systems in Compositional Languages for Parallel Programming. – Manuscript.

Candidate's thesis on Physics and Mathematics, speciality 01.05.03 – Mathematics and Software for Computers and Systems. – Taras Shevchenko National University of Kyiv, – Kyiv, 2019.

Parallel environments today are widespread, since single-processor systems have been significantly restricted in performance. For many modern systems, the need to prove their properties, in particular, the correctness of functioning, is urgent. One of the largest subclasses of such parallel programs is client-server systems. Therefore, the question of constructing adequate models of parallel systems, including such with a shared memory, allows to effectively perform mathematical verification of the properties of these systems. The thesis is aimed at reducing the gap between existing theoretical models and real parallel systems by expanding and refining such models. The basic model of the parallel systems under investigation is the implementation of shared memory interleaving concurrency. This is an adequate model of server systems for most client-server complexes, in particular, service-oriented, network and other software systems.

Over the years, several approaches to this problem have been developed, in particular: insertion programming (in the works of Academician Letichevsky O.A. and others), a number of models and approaches developed by C.A.R. Hoare, and developed by other authors, which is discussed in detail in section 2 of the dissertation. These are actively investigated in the Institute of Cybernetics and the Institute of Software Systems of the National Academy of Sciences of Ukraine, as well as the Department of Theory and Programming Technology of the Faculty of Computer Science and Cybernetics of the Taras Shevchenko National University of Kyiv is paying considerable attention to these problems. All this confirms the relevance of the research. In addition, at the Department of Theory and Technology of Programming academician Redko V.N. and prof. Nikitchenko M.S. developed the original compositional approach to programming, which relies primarily on the semantics of programming languages and has advantages over other formal methods.

The urgency of the topic of the dissertation is due to the fact that the client-server systems are used very widely in different fields, therefore, in many cases, a guarantee of the performance of some properties of systems is required. At the same time, parallelism, inextricably linked with the nature of such systems, adds complexity to the model, hence the development of formal models and systems that are adequate to the client-server environments, while at the same time easy-to-use - for the next proof of the properties of such programs - is important and relevant the subject of computer science.

This determines the relevance of the topic of the dissertation, which is devoted to the problems of constructing adequate models of software systems, in particular server environments, in order to prove the correctness of the software by specification and verification. The use of composition-nominative methods for solving these problems

allows enriching the specifications of the instrument and expanding the possibilities of verification for the tasks studied in the work.

The proposed extension of the IPCL language and execution model (including the state of the program) is a more adequate representation for parallel systems with a dynamic generation of programs (the most commonly used in programming). On the basis of the specification of the mechanism of generating parallel programs, a new compositional-nominative model was constructed and the adequacy of the server environment of the classical client-server systems, as well as the pragmatic completeness of this model, was substantiated. A class of composition-nominative languages with compositions of dynamically starting (spawning) and joining of parallel programs during execution in runtime has been constructed and studied, as well as a number of basic functions in these languages are presented (atomic and others). For the constructed model it is shown, under which conditions it will be equal with the class of languages IPCL. The method of property verification in the class of server programs in the composited languages is adapted. A theorem is formulated and proved, which defines the conditions when the extension of the model is equivalent to the existing model of multi-instance execution in IPCL.

The scientific novelty is as follows: new expansion of compositional languages of specifications has been constructed and investigated, which enables to adequately model client-server systems with dynamic generation of programs; an analysis of the approaches to the generation of new programs in a parallel environment was conducted, their models in the compositional approach were constructed; on the basis of the clarification of the mechanism of generating parallel programs, a new composition-nominative model was constructed and the adequacy of the server environment of the classical client-server systems, as well as the pragmatic completeness of this model, was substantiated; a class of composition-nominative languages with compositions of spawning (start) and joining (join) of parallel programs during execution is constructed and investigated, as well as a number of basic functions in these languages having been presented (atomic and others); for the constructed model it is shown, under what conditions it will be equal with the class of languages IPCL; the method for verification of properties in the class of server programs in the composited languages has been adapted.

The dissertation has a theoretical and applied orientation. Work results can be used to specify and verify parallel systems with a dynamic generation of instances of programs that interact via shared memory. The obtained results were introduced into the educational process of the educational program "Informatics" of 122 specialization "Computer Science" at the Faculty of Computer Science and Cybernetics of the Taras Shevchenko National University of Kyiv, and also applied to prove the properties of real-world client-server software systems.

Key words: compositional programming, composition-nominative languages, program's properties proof, formal methods for verification, client-server systems, parallel programs, shared memory concurrency.