ХЕРСОНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

Кваліфікаційна наукова
праця на правах рукопису

ТАРАСІЧ ЮЛІЯ ГЕННАДІЇВНА

УДК 004.415:519.7:004.942(043.3)

ДИСЕРТАЦІЯ

СТАТИЧНИЙ АНАЛІЗ ЛІНІЙНО ВИЗНАЧЕНИХ ПРОГРАМ І ЙОГО ЗАСТОСУВАННЯ

121 Інженерія програмного забезпечення

12 Інформаційні технології

Подається на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело
_____ Ю.Г. Тарасіч
Науковий керівник _____Львов Михайло Сергійович, доктор фізико-математичних наук, професор

Херсон - 2021

АНОТАЦІЯ

Тарасіч Ю.Г. Статичний аналіз лінійно визначених програм і його застосування. - Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 12.121 Інженерія програмного забезпечення. – Херсонський державний університет, Херсон, 2020.

Загальна проблема дослідження полягає у дослідженні та розробці методів створення моделей ефективних програмних систем статичного аналізу програм та математичних моделей об'єктних програм для статичного аналізу, дослідженні математичних методів та розробці ефективних алгоритмів комп'ютерної алгебри статичного аналізу програм.

Підхід проведення досліджень полягає у:

- аналізі методів та функціональності існуючих систем статичного аналізу програм,
- теоретичному аналізі результатів досліджень з теорії програмних інваріантів, що включає аналіз математичних моделей програм та алгоритмів комп'ютерної алгебри що використовуються у теорії програмних інваріантів,
- створенні нових алгоритмів комп'ютерної алгебри.

Мета роботи полягає в аналізі існуючих алгоритмів комп'ютерної алгебри та інсерційного моделювання в системах статичного аналізу та верифікації програмного забезпечення та побудові ефективних алгоритмів статичного аналізу програм.

Основні завдання дослідження:

- Дослідження існуючих спеціалізованих систем статичного аналізу програм.
- Аналіз алгоритмів комп'ютерної алгебри та інсерційного моделювання в системах статичного аналізу та верифікації програмного забезпечення.
- Аналіз існуючих алгоритмів побудови канонічних форм логічних формул та інструментів спрощення формул (солверів).

- Створення нових засобів та алгоритмів статичного аналізу програм, зокрема алгоритмів доказу інваріантних нерівностей.

- Розробка нових алгоритмів побудови канонічних форм лінійних напівалгебраїчних формул.

- Реалізація алгоритму побудови канонічних форм лінійних напівалгебраїчних формул.

У першому розділі дисертації (*Статичний аналіз лінійно визначених програм*) надано аналіз існуючих методів та систем статичного аналізу, приведено базисні дефініції дисертаційного дослідження – визначено загальні поняття графових та алгебраїчних моделей програм, приведено визначення поняття лінійно визначених програм.

У другому розділі дисертаційного дослідження (*Генерація інваріантів програм*) приведено поняття власного полінома лінійного оператора та алгоритм побудови власних поліномів, визначено зв’язок між власними поліномами лінійних операторів та поліноміальними інваріантами лінійних циклів програм (підрозділ 2.1), представлено нові методи доказу інваріантності системи лінійних нерівностей і завершуваності лінійно визначених ітеративних циклів імперативних програм (підрозділ 2.2).

Основні алгоритми проблеми доказу та генерації поліноміальних інваріантних рівностей лінійно визначених циклів - побудова множини L – інваріантів циклів для операторів, жорданова форма яких містить нетривіальні блоки; теорема 2.12, в якій встановлено структуру базису Гребнера ідеалу поліноміальних інваріантів лінійного циклу з довільним невироджених лінійним оператором в тілі циклу і теорема 2.14 про алгоритмічну розвязуваність проблеми побудови базису ідеалу інваріантів «діагоналізуємої частини» лінійного оператора в разі неприводимості його мінімального характеристичного полінома. У зв'язку з цим, за теоремою 2.12, інваріанти лінійного оператора можна класифікувати як внутрішньоклітинкові - ті, які притаманні кожній жордановій клітинці лінійного оператора, і міжклітинкові - ті, які притаманні його діагоналізуємій

частині. Внутрішньоклітинкові інваріанти обчислюються безпосередньо за формулами $x_j = \frac{z}{c}(a_j + \frac{C_1(\lambda\frac{cy-bz}{cz})}{\lambda}a_{j+1} + ... + \frac{C_{n-j}(\lambda\frac{cy-bz}{cz})}{\lambda^{n-j}}a_n)$. Існування міжклітинкових інваріантів залежить від існування нетривіальних мультиплікативних співвідношень між власними числами лінійного оператора (теорема 2.2).

Для лінійних операторів з непривідним мінімальним характеристичним многочленом проблема побудови базису множини мультиплікативних співвідношень між його власними числами алгоритмічно розв'язна, але алгоритм теореми 2.14 є неефективним зважаючи на дуже великі степені многочлена $S(x)$, який потрібно розкладати на множники.

У методі доказу інваріантності системи лінійних нерівностей, а також завершуваності лінійно визначених ітеративних циклів імперативних програм враховується передумова циклу та умова його повторення у вигляді сукупності системи лінійних нерівностей. В основі методу лежить побудова та аналіз спектру лінійного оператора тіла циклу та обчислення числа його ітерацій, після виконання яких інваріантність або підтверджується або спростовується.

Запропоновані методи доказу та генерації інваріантних рівностей та доказу інваріантних нерівностей може бути покладено в основу загального алгоритму доказу інваріантності системи лінійних нерівностей та поліноміальних рівностей для лінійно-визначених програм.

У розділі 3 (*Канонічні форми лінійних напівалгебраїчних формул над типами даних та їх використання в задачах верифікації програм*) представлено результати аналізу функціональних особливостей існуючих систем побудови доказів (на прикладі CVC4, MathSAT5, QEPCAD, Singular, COCOA, MiniZinc, STP, RedLog, Satallax, Isabelle, E-SETHEO, Minisat, SMTInterpol, TPS / ETPS, Paradox, Gandalf, Vampire), зокрема на наявність інструментів спрощення формул (підрозділ 3.1); представлено новий

алгоритм побудови канонічної форми лінійних півалгебраїчних (ЛПФ) формул (підрозділи 3.2-3.3), включаючи алгоритми побудови канонічних форм логічних формул над перелічуваним та множиним типами даних (підрозділ 3.4); представлено модифікацію алгоритму побудови канонічної форми лінійних напівалгебраїчних формул за рахунок застосування алгоритму поповнення (підрозділ 3.6), приведено реалізацію алгоритму побудови канонічної форми ЛПФ засобами IMS та C++ (підрозділ 3.5).

Основний результатом є визначення канонічної форми ЛПФ, що володіє властивістю єдиності і іншими корисними властивостями, а також представлення алгоритму її побудови. Пропонована канонічна форма ЛПФ - пряме узагальнення канонічної форми представлення системи лінійних нерівностей і алгоритму її побудови, наведених у [3.37, 3.38].

Наукова новизна одержаних результатів.

— Уперше представлено новий метод визначення канонічної форми ЛПФ, що володіє властивістю єдиності та іншими корисними властивостями, а також алгоритм її побудови.

— Уперше реалізовано алгоритм побудови канонічних форм логічних формул засобами інсерційного моделювання (система інсерційного моделювання IMS).

— Набуло подальшого розвитку дослідження проблеми пошуку інваріантів циклів, а саме, - наведено алгоритм обчислення основних інваріантів лінійного оператора жорданової клітинки та алгоритм обчислення основних інваріантів діагоаналізуємого лінійного оператора з непривідним мінімальним характеристичним поліномом та представлено новий метод доказу інваріантності системи лінійних нерівностей і завершення деяких лінійних ітераційних циклів імперативних програм, дані яких є елементами конструктивного лінійно впорядкованого поля.

Практичне значення наукових результатів полягає в можливості застосування наукових положень і висновків дослідження в задачах верифікації програмного забезпечення, а саме – в розробці спеціалізованих

програмних систем верифікації формальних моделей програм.

Розвиток даного підходу у верифікації дозволить досить ефективно та надійно розробляти прогамне забезпечення для медичної апаратури, військової техніки, бортових систем управління для авіації та космонавтики, технології блокчейн, в загалому, критичних програмних систем.

На теперішньому етапі виконується дослідження застосування методів формальної верифікації, зокрема методів статичного аналізу, та використання систем верифікації для верифікації формальних моделей програм, економічних, біологічних процесів, нормативно-правових документів, тощо, продовжується робота над дослідженням математичних методів та розробкою ефективних алгоритмів комп'ютерної алгебри статичного аналізу програм. Відповідні результати представлено у [218-220].

Ключові слова: статичний аналіз, верифікація, лінійно визначені програми, інваріант циклу, програмні інваріанти

ABSTRACT

Yuliia H. Tarasich. **The static analysis of linearly defined programs and its application. - Qualification scientific paper, manuscript.**

Thesis of a Doctor of Phylosophy in specialty 12.121 Software engineering. - Kherson State University, Kherson, 2020.

The general problem of study is research and development of methods of creation of models of effective software systems for static programs analysis and mathematical models of object programs for static analysis, research of mathematical methods and development of effective algorithms of computer algebra of programs static analysis.

The research approach consist of:

- the analysis of methods and of functionality of existing systems of static programs analysis,

- the theoretical analysis of research results in the theory of program invariants, including analysis of mathematical models of programs and computer algebra algorithms that are used in the theory of program invariants,

- the development of new computer algebra algorithms.

The **main aim of the work** is to analyze the existing algorithms of computer algebra and insertion modeling in the systems of software static analysis and verification and to build an effective algorithms for static analysis of programs.

The main tasks of the study**:**

- Research of existing specialized systems for static analysis of programs.

- Analysis of algorithms of computer algebra and insertion modeling in static analysis and software verification systems.

- Analysis of existing algorithms for constructing canonical forms of logical formulas and simplification tools (solvers).

- Creation of new tools and algorithms for static analysis of programs, in particular, creation of algorithms for proving of invariant inequalities.

- Development of new algorithms for constructing canonical forms of linear semi-algebraic formulas.

- Implementation of the algorithm for constructing canonical forms of linear semi-algebraic formulas.

In the first section of the dissertation (*The static analysis of linearly defined programs*) an analysis of existing methods and systems of static analysis and basic definitions of dissertation research (such as general concepts of graph and algebraic models of programs, the concept of linearly defined programs) are provided.

In the second section of the dissertation research (*Generation of invariants of programs*) the concept of eigenpolynom of a linear operator is introduced, the algorithm of construction of eigenpolynomials is formulated and the connection between eigenpolynoms and polynomial invariants of linear cycles of programs is established (subsection 2.1); a new method for proving the invariance of the system of linear inequalities, and also of the completion of linearly defined iterative cycles of imperative programs is presented (subsection 2.2).

The main algorithms of the problem of proof and generation of polynomial invariant equations are the construction of the set L - invariants of cycles for operators whose Jordan form contains nontrivial blocks; Theorem 2.13, in which the structure of the Grebner basis of the ideal of polynomial invariants of a linear cycle with an arbitrary nondegenerate linear operator in the body of the cycle is establishing and Theorem 2.15 about the algorithmic solvability of the problem of constructing the basis of the ideal of invariants of the "diagonalizable part" of a linear operator in the case of irreducibility of its minimal characteristic polynomial. Thus, by theorem 2.13, the invariants of a linear operator can be classified as intracellular - that are inherent to each Jordan cell of linear operator, and intercellular - those that are inherent in its diagonalisable part.

Intracellular invariants are computed directly from the formulas $x_j = \frac{z}{c}(a_j + \frac{C_1(\lambda\frac{cy-bz}{cz})}{\lambda}a_{j+1} + ... + \frac{C_{n-j}(\lambda\frac{cy-bz}{cz})}{\lambda^{n-j}}a_n)$. The existence of intercellular

invariants depend on the existence of nontrivial multiplicative relations between the eigenvalues of the linear operator (theorem 2). For linear operators with an irreducible minimum characteristic polynomial problem of constructing a basis of set of multiplicative relations between its eigenvalues is algorithmically solvable, but the algorithm of theorem 8 is ineffective due to a very large degree of the polynomial $S(x)$, which is be necessary to decompose into factors.

The method of proving the invariance of the system of linear inequalities, and and also of the completion of linearly defined iterative cycles of imperative programs takes into account the premise of the cycle, and also the condition of cycle repetition in the form of a set of systems of linear inequalities. The method is based on the construction and analysis of the spectrum of the cycle body and on the calculation of the number of iterations of the cycle, after which the invariance is confirmed or refuted.

The proposed methods of proof and generation of invariant equations and proof of invariant inequalities can be used as a basis for the general algorithm for proving the invariance of the system of linear inequalities and polynomial equalities for linearly defined programs.

In Section 3 (*Canonical forms of linear semi-algebraic formulas over the data types and their use in program verification*) the results of the analysis of functional features of existing proving systems, in particular on the availability of formula simplification tools, (CVC4, MathSAT5, QEPCAD, Singular, COCOA, MiniZinc, STP, RedLog, Satallax Isabelle, E-SETHEO, Minisat, SMTInterpol, TPS / ETPS, Paradox, Gandalf, Vampire) are presented (section 3.1); a new algorithm for constructing the canonical form of linear semi-algebraic (LSF) formulas (subsections 3.2-3.3) is presented, including algorithms for constructing canonical forms of logical formulas over enumerated and multiple types (subsection 3.4); the modification of the algorithm for constructing the canonical form of linear semi-algebraic formulas due to the application of the replenishment algorithm is presented (subsection 3.6), the implementation of the algorithm for constructing the canonical form of LSF by means of IMS and C ++ (subsection 3.5) is given.

The main result is the definition of the canonical form of LSF, which has the property of unity and other useful properties, and also the description of the algorithm for its construction. The proposed canonical form of LSF is a direct generalization of the canonical form of representation of the system of linear inequalities and the algorithm of its construction that were given in [3.37, 3.38].

Scientific novelty of the obtained results:

- the problem of generating of polynomial invariants of iterative cycles with a loop initialization operator and a nondegenerate linear operator in the loop body is considered.

- the algorithm of calculation of the basic invariants of the linear operator of a Jordan cell and the algorithm of calculation of the main invariants of the diagoanalyzed linear operator with the irreducible minimum characteristic polynomial are shown.

- the new method of proving the invariance of a system of linear inequalities and the completion of some linear iterative cycles of imperative programs, the data of which are elements of a constructive linearly ordered field, is presented.

- a new method of determining the canonical form of LSF, which has the property of uniqueness and other useful properties, and also the description of the algorithm for its construction were given.

- the algorithm of construction of canonical forms of logical formulas is realized.

The practical significance of the expected scientific results lies in the possibility of applying scientific provisions and research conclusions in software verification tasks, namely in the development of specialized software systems for verification of formal program models.

The development of this approach in verification will allow to effectively and reliably develop a medical equipment, a military equipment, an onboard control systems for aviation and aerospace, a blockchain technology, in general, the software, where the cost of error and unreliability of software can be expensive.

At the present stage we work on the study of the application of formal

verification methods, including methods of static analysis, and of the using of verification systems to verify formal models of programs, economic, biological processes, legal documents, etc. We also continue to work on the study of mathematical methods and of the development of efficient algorithms of computer algebra of static program analysis. The corresponding results are presented in [218-220].

Keywords: static analysis, verification, linearly defined programs, cycle invariant, program invariants.

СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Основні наукові результати дисертації

1. Lvov M. The Static Analysis of Linear Loops [Електронний ресурс] / M. Lvov, Y. Tarasich // CEUR Workshop Proceedings. – 2015. – Режим доступу до ресурсу: http://ceur-ws.org/Vol-1356/paper_79.pdf.

Особистий внесок здобувачки полягає в аналізі та описі основних результатів проблеми доказу та генерації поліноміальних інваріантних рівностей, визначенні нових засобів та алгоритмів статичного аналізу програм, а саме - нового алгоритму доказу інваріантних нерівностей.

2. The canonical forms of logical formulas off the data types / M. Lvov, V. Peschanenko, O. Letychevskyi, Yu. Tarasich // 13th International Conference Theoretical and Applied Aspect of Program Systems Development. – 2016. (тези)

Особистий внесок здобувачки полягає у представлені нових алгоритмів обчислення канонічної форми логічних формул над перелічуваним та множинним типами даних.

3. The canonical forms of logical formulae over the data types and their using in programs verification [Електронний ресурс] / M.Lvov, V. Peschanenko, O. Letychevskyi, Y. Tarasich // CEUR Workshop Proceedings. – 2017. – Режим доступу до ресурсу: http://ceur-ws.org/Vol-1844/10000536.pdf.

Особистий внесок здобувачки полягає у стислому огляді засобів спрощення формул, представлені нового алгоритму обчислення канонічної форми логічних формул над упорядкованими типами даних.

4. Algorithm and Tools for Constructing Canonical Forms of Linear Semi-Algebraic Formulas / [M. Lvov, V. Peschanenko, Y. Tarasich and all.]. // Cybernetics and Systems Analysis. – 2018. – №54. – С. 993–1002.

Особистий внесок здобувачки полягає в тестуванні засобів спрощення формул та аналізі їх основних функціональних особливостей; представленні

нового алгоритму побудови канонічних форм лінійних напівалгебраїчних формул.

Апробація матеріалів дисертації

5. Formalization and algebraic modeling of tokenomics projects [Електронний ресурс] / [O. Letychevskyi, V. Peschanenko, V. Radchenko та ін.] // CEUR Workshop Proceedings. – 2019. – Режим доступу до ресурсу: http://ceur-ws.org/Vol-2393/paper_409.pdf.

Особистий внесок здобувачки полягає у застосуванні розроблених алгоритмів для моделювання визначених задач.

6. Our Approach to Formal Verification of Token Economy Models / O.Letychevskyi, V. Peschanenko, M. Poltoratskyi, Y. Tarasich, 2020. – (Communications in Computer and Information Science). – (1175 CCIS). – С. 348 – 363.

Особистий внесок здобувачки полягає у застосуванні розроблених алгоритмів для моделювання визначених задач.

7. Platform for Modeling of Algebraic Behavior: Experience and Conclusions / Letychevskyi, O., Peschanenko, V., Poltoratskyi, M., Tarasich, Y...// CEUR Workshop Proceedings. - 2020. – Режим доступу до ресурсу: http://ceur-ws.org/Vol-2732/20200042.pdf

Особистий внесок здобувачки полягає у застосуванні розроблених алгоритмів для моделювання визначених задач.

ЗМІСТ

ВСТУП

На сьогодні інформаційні технології займають провідні позиції майже у всіх сферах ведення бізнесу. Потреби сучасної промисловості провокують зростання складності програмних систем. Для вирішення бізнес-задач створюються великі, розподілені, інформаційні системи, які постійно модифікуються та мають тенденцію до ускладнення. Вони можуть мати в своєму складі як готові додатки, так і зовнішні IT-сервіси (SaaS), власні програмні ресурси та ресурси розроблені за замовленням, безкоштовні продукти з відкритим вихідним кодом (open source). Проблеми в їх роботі призводять до порушення інформаційної безпеки, і як наслідок, до фінансових і репутаційних втрат бізнесу. Так, за даними дослідження [1], проведеному Ponemon Institute за підтримки Hewlett Packard Enterprise, в останні роки фінансові втрати бізнесу значно збільшилися через кібератаки. До наслідків кібератак дослідники віднесли крадіжку інтелектуальної власності організацій, конфіскацію банківських рахунків в інтернеті, створення та розповсюдження вірусів, порушення критичної національної інфраструктури країн, тощо.

Помилки у програмному забезпеченні (ПЗ) мають значний вплив на його якість, приводячи до збоїв в його роботі, а в деяких випадках (уразливості) – до перехвату контролю над системою. Зважаючи на значні збільшення розмірів програм, за останні 20 років, значно зросла і складність пошуку помилок у коді. Відповідно, постає гостра потреба у постійному вдосконаленні методів тестування, випробувань та контроля якості програмного забезпечення.

На сьогодні запропоновано безліч методів пошуку помилок. Їх використання, у свою чергу, регламентується створеними стандартами та циклами безпечної розробки програмного забезпечення, наприклад, Microsoft Security Development Lifecycle [2], ГОСТ Р 56939-2016 «Захист інформації. Розробка безпечного програмного забезпечення. Загальні вимоги» [3], IEEE

829 (документація тестування ПЗ) [4], IEEE 1008 (модульне тестування ПЗ) [5], групи стандартів ISO 14598 [6-11] (описують процеси оцінки програмних систем і компонентів, засновані на визначенні метрик і показників, пов'язаних з певними характеристиками якості), ISO 15504 [12-15] (описує метод SPICE оцінки та вдосконалення процесів розробки програмного забезпечення), IEEE 1028 [16] (описує один з методів проведення верифікації). Основним стандартом, що регулює планування і проведення верифікації ПЗ, є стандарт IEEE 1 012 на процеси верифікації та валідації [17,18].

Методи тестування не забезпечують вичерпного аналізу всіх можливих варіантів поведінки систем, тим самим, не можуть гарантувати відсутність порушення властивостей, якими повинні володіти системи, що розробляються. Від 40 до 75% трудових ресурсів промисловості витрачаються на тестування і валідаціюпрограмного забезпечення [19, 20], а частка дефектів у вимогах, виявлених на стадії тестування, становить 40-50% [21]. Основний відсоток помилок у програмних засобах виявляється на етапах тестування та експлуатації. Поява помилок на етапі експлуатації часто призводить до катастрофічних наслідків, тому актуальною задачею є виявлення помилок на більш ранніх етапах розробки ПЗ. Відповідно, розробка систем зі складною моделлю поведінки має включати такий обовязковий елемент, як автоматизація перевірки правильності ПЗ – верифікацію. Верифікацією називається перевірка властивостей формального опису моделі.

Формальні методи верифікації ПЗ використовують формальні моделі вимог, поведінки і оточення ПЗ для аналізу його властивостей. Такі моделі є або логіко-алгебраїчними, або виконуваними, або проміжними, що мають риси і логіко-алгебраїчних, і виконуваних моделей.

В якості математичних об'єктів моделювання систем часто використовуються транзиційні системи, мережі Петрі, скінчені, часові та гібридні автомати, алгебри процесів і ін. У якості прикладу властивостей що

перевіряються можна назвати властивості живучості, безпеки, повноти і несуперечності. У роботі Хоара сформульовано [22] найбільш грандіозне завдання верифікації програм, відоме як «Grand challenge» - створення верифікуючого компілятора. Розрізняють два основні підходи до формальної верифікації: логічние виведення (logical inference) і перевірка моделей (model checking). Теорії і техніці верифікації, об'єднаної назвою "model checking", присвячено велику кількість наукових праць, проводяться спеціалізовані конференції. Останнім часом метод широко використовується для перевірки властивостей повноти, несуперечності і безпеки як програмного, так і апаратного забезпечення промислових систем.

Статична верифікація програм полягає в аналізі коду програми без її реального виконання, що на відміну від динамічного аналізу не вимагає налаштування тестового оточення і надає можливість проаналізувати всі можливі виконання програми, навіть ті, для відтворення яких потрібно одночасне виконання відразу декількох умов, які рідко зустрічаються.

Одним з основних методів пошуку помилок, обов'язковим до застосування, є статичний аналіз вихідного коду програм.

Основною задачею статичного аналізу програм є задача доведення коректності програми за її висхідними кодом. Доведення має спиратися на так звані інваріанти, задачі автоматичного доведення та пошуку яких є ключовою проблемою аналізу програм.

Перевагами методу статичного аналізу є:

- одночасний аналіз багатьох шляхів виконання (масштабованість),
- пошук помилок на шляхах, що рідко виконуються (погано покриваються тестуванням або динамічним аналізом).

Останнім часом усе більшого поширення набуває застосування статичниого аналізу для верифікації властивостей ПЗ,яке використовується в комп'ютерних системах високої надійності, особливо критичних для життя (Safety-critical): для пошуку коду з потенційними уразливостями(Static Application Security Testing, SAST). Зокрема застосування статичного аналізу

програм залишається важливим та актуальним в таких областях, як верифікація ПЗ для медичних пристроїв, систем захисту реакторів (Reactor Protection Systems), ядерних станцій ,ПЗ для авіації та військової техніки, тощо [23 - 26].

Зважаючи на швидкі темпи росту популярності та поширення блокчейн технології, реалізації концепції смарт-контракту, особливої уваги потребує і верифікація відповідних програмних систем та моделей. Так, лише за першу половину 2018 року кількість смарт-контрактів у блокчейні Ethereum виросла вдвічі, у порівнянні з попереднім роком, а, відповідно, зросла і кількість вразливостей, векторів атак на децентралізовані додатки. Як приклад, вартість втрат бізнесу внаслідок атак склала 30 млн. доларів для Parity та 53 млн. доларів для DAO [27].

Загальна проблема дослідження полягає у дослідженні та розробці методів створення моделей ефективних програмних систем статичного аналізу програм та математичних моделей об’єктних програм для статичного аналізу, дослідженні математичних методів та розробці ефективних алгоритмів комп’ютерної алгебри статичного аналізу програм.

Підхід проведення досліджень полягає у:

- аналізі методів та функціональності існуючих систем статичного аналізу програм,
- теоретичному аналізі результатів досліджень з теорії програмних інваріантів, що включає аналіз математичних моделей програм та алгоритмів комп’ютерної алгебри що використовуються у теорії програмних інваріантів,
- створенні нових алгоритмів комп’ютерної алгебри.

Об'єкт дослідження: Спеціалізовані програмні системи статичного аналізу та верифікації програмного забезпечення.

Предмет дослідження: Алгоритми статичного аналізу лінійно визначених програм та їх застосування.

Мета роботи полягає в аналізі існуючих алгоритмів комп’ютерної алгебри та інсерційного моделювання в системах статичного аналізу та

верифікації програмного забезпечення та побудові ефективних алгоритмів статичного аналізу програм.

Основні завдання дослідження:

- Дослідження існуючих спеціалізованих систем статичного аналізу програм.

- Аналіз алгоритмів комп'ютерної алгебри та інсерційного моделювання в системах статичного аналізу та верифікації програмного забезпечення.

- Аналіз існуючих алгоритмів побудови канонічних форм логічних формул та інструментів спрощення (солверів).

- Створення нових засобів та алгоритмів статичного аналізу програм, зокрема алгоритмів доказу інваріантних нерівностей.

- Розробка нових алгоритмів побудови канонічних форм лінійних напівалгебраїчних формул.

- Реалізація алгоритму побудови канонічних форм лінійних напівалгебраїчних формул.

Для досягнення поставленої мети та розв'язання визначених завдань у дисертаційній роботі використано такі методи наукових досліджень:

— систематизації, порівняння та узагальнення – аналіз систем статичного аналізу програм (підрозділ 1.2 розділу 1), аналіз методів та технік статичного аналізу (підрозділ 1.2 розділу 1), істрорична ретроспектива досліджень проблеми потокового аналізу програм (2 розділ), аналіз інструментів побудови доказів/спрощувачів (підрозділ 3.1 розділу 3).

— положення та методи математичної логіки, методи теорії алгоритмів, теорії алгоритмічних систем, теорії програмування, методи теорії графів, теорії формальних мов та граматик, теорії програмних інваріантів, методи компютерної алгебри (підрозділи 2.1, 2.2 розділу 2, підрозділи 3.2-3.4, 3.6 розділу 3).

— методи теорії алгебраїчного програмування, теорії інсерційного моделювання – реалізація алгоритму побудови канонічних форм лінійних напівалгебраїчних формул (підрозділ 3.5 розділу 3).

Теоретичною, методологічною та інформаційною основою дисертаційної роботи є наукові праці вітчизняних та зарубіжних авторів, матеріали періодичних видань, огляди, результати експертних досліджень, Інтернет-ресурси.

Наукова новизна одержаних результатів.

— Уперше представлено новий метод визначення канонічної форми ЛПФ, що володіє властивістю єдиності та іншими корисними властивостями, а також алгоритм її побудови.

— Уперше реалізовано алгоритм побудови канонічних форм логічних формул засобами інсерційного моделювання (система інсерційного моделювання IMS).

— Набуло подальшого розвитку дослідження проблеми пошуку інваріантів циклів, а саме, - наведено алгоритм обчислення основних інваріантів лінійного оператора жорданової клітинки та алгоритм обчислення основних інваріантів діагоаналізуємого лінійного оператора з непривідним мінімальним характеристичним поліномом та представлено новий метод доказу інваріантності системи лінійних нерівностей і завершення деяких лінійних ітераційних циклів імперативних програм, дані яких є елементами конструктивного лінійно впорядкованого поля.

За результатами дослідження опубліковано 7 наукових праць загальним обсягом 97 ст. (з них 24,5 належать особисто автору) – 6 наукових статей, у тому числі 6 у фахових виданнях та виданнях, що індексуються Scopus та 1 - тези доповіді на науковій конференції.

Апробація матеріалів дисертації: Міжнародна наукова конференція ICTERI, м. Львів, травень 2015 р., Міжнародна наукова конференція Теоретичні та прикладні аспекти побудови програмних систем, м. Київ, 5-9 грудня 2016 р., Міжнародна наукова конференція сучасна інформатика: проблеми, досягнення та перспективи розвитку, м. Київ, 13-15 грудня, 2017 р., Міжнародна наукова конференція ICTERI, м. Київ, 15-19 травня 2017 р.,

Міжнародна наукова конференція ICTERI, м. Херсон, травень 2019 р., Міжнародна наукова конференція ICTERI, м. Харків, жовтень 2020 р.

Дисертаційна робота складається із вступу, основної частини, що містить три розділи, висновків, списку використаних джерел, додатків. Повний обсяг дисертаційної роботи – 175 сторінок тексту, у тому числі - 8 сторінок додатків, 37 сторінок списку використаних джерел (8 сторінок – до першого розділу, 6 – до другого, 5 сторінок – до третього розділу. 18 сторінок – загальний список використаних джерел) у кількості 220 найменувань (загальний список).

Зв'язок роботи з науковими планами та темами. Держбюджетна науково-дослідна робота № 4/16-18 “Методи комп’ютерної алгебри та інсерційного моделювання в системах статичного аналізу і верифікації програмного забезпечення” - 0115U001128. Термін виконання - 2016-2018 рр.

Практичне значення очікуваних наукових результатів полягає в можливості застосування наукових положень і висновків дослідження в задачах верифікації програмного забезпечення, а саме – в розробці спеціалізованих програмних систем верифікації формальних моделей програм.

Використання системи інсерційного моделювання, заснованої на транзиційних системах, для яких також створено графічний редактор на базі платформи Eclipse, та імплементація результатів дослідження до відповідних систем дозволить досліджувати моделі із необмеженими станами. Навідміну від підходу до аналізу програм, запропонованого у [28] ми отримаємо алгоритм перевірки control-flow графу програм, що дозволить виявити значно більшу кількість помилок.

Відповідно, розвиток даного підходу у верифікації дозволить більш ефективно та надійно розробляти медичну апаратуру, військову техніку, бортові системи управління для авіації та космонавтики, технології блокчейн, в загалому там де ціна помилки та ненадійності програмного забезпечення може дорого коштувати.

Методи формальної верифікації можуть також бути застосовані не тільки для покращення якості програмного забезпечення, але й для доведення ефективності, пошуку шляхів покращення для будь-яких бізнес-процесів, економічних моделей, представлених у формалізованому вигляді вигляді – математичними моделями. Введення багаторівневих середовищ у систему інсерційного моделювання, дозволить розширити можливості системи для побудови більш складних, ієрархічно організованих моделей.

Імплементація результатів дослідження у курси Теорія аналізу програм; Комп'ютерна алгебра; Моделювання та проєктування програмних систем навчального призначення; Формальні методи специфікації, верифікації та оптимізації програм; Системи штучного інтелекту; Спеціальні мови програмування, ш т. ін., що викладаються для студентів ІТ спеціальностей надасть змогу значно поглибити знання студентів в області верифікації ПЗ.

РОЗДІЛ 1. СТАТИЧНИЙ АНАЛІЗ ЛІНІЙНО ВИЗНАЧЕНИХ ПРОГРАМ

1.1. Програми інтерпретовані над алгебрами даних. Лінійно визначені програми.

У теорії програмування об'єкт дослідження, як правило, представлено математичною моделлю програми. В [1] відповідним моделям відповідають операторні схеми програм, схеми програм Янова. В теорії аналізу та перетворення програм використовуються або графові [2-4], або алгебраїчні моделі програм [5-7]. До графових моделей відносяться, наприклад, $U-Y$ схеми програм [2, 9], транзиційні системи (transition systems) [10].

Класичним прикладом графової моделі програми є її блок-схема. У сучасних дослідженнях, математична модель програми подається у вигляді графу. Стани, в яких знаходиться програма у різні проміжки часу (контрольні точки), позначають вершини графа, а дії, які виконує програма – ребра графа з мітками. Переходи від одного стану до іншого відображають виконання програми у дискретному часі.

Алгебраїчними моделями є алгоритмічні алгебри В.М.Глушкова, терми, що визначаються в програмних логіках.

Адекватні способи представлення моделей програм у алгебраїчній формі представлено у роботах Т. Хоара, Р. Мілнера, Й. Баетена, Л. Карделлі, Дж. Хіллстона, В.М. Глушкова, О.А. Летичевського, Ю.В. Капітонової, М.С. Нікітченка, М.С. Львова, В.П. Клименка), експоненціальний вибух кількості станів моделі досліджено у працях Е. Кларка, К. Макмілана, О.А. Летичевського, нерозв’язність задачі досяжності стану у моделі відображено в роботах К. Геделя, А. Тьюрінга, Е. Поста, С. Кліні, А. Черча.

В залежності від рівня абстракції та різних рівнів деталізації операцій, для одного й того ж програмного засобу може бути створено різні моделі.

Дослідження передбачає використання як графових, так і алгебраїчних моделей програм.

Визначимо загальні поняття для цих моделей [11]:

- Пам'ять X програми P є скінченою множиною (вектором) змінних $X = (x_1,...,x_n)$;
- Алгебра даних D - область визначення всіх змінних (пам'яті) програми. Алгебра даних є багатосортною алгеброю та визначає множину операцій над даними [11]. Зокрема, визначається стандартний сорт D - сорт *Boolean* та ієрархія сортів, над яким за допомогою операторів присвоювання визначено обчислення.
- Стан $\bar{d}$ пам'яті програми P - вектор $\bar{d} = (d_1,...,d_n)$. Якщо пам'ять програми перебуває в стані $\bar{d}$, це означає, що $x_1 = d_1,...,x_n = d_n$. Таким чином, ми визначаємо простір станів пам'яті як множину D^X [11];
- Оператори присвоєння інтерпретуються як векторні. Оператор $y = (x_1 := F_1(X),...,x_n := F_n(X))$ перетворює простір станів пам'яті: $y : D^X \to D^X$. Це означає, що компоненти оператора присвоювання виконуються одночасно (паралельно). Множину операторів присвоєння позначимо через Y. Відзначимо, що в деяких задачах аналізу програм варто розглядати послідовне виконання компонентів [11];
- Множина умов - U.Умови перетворюють терм F алгебри D в (*True, False*).

$U - Y$ - *програми:*

$U - Y$ – схемою програми над пам'яттю ($U - Y$ – програмою) називають ініціальний зв'язний граф P з виділеною підмножиною заключних вершин (станів), дуги якого відмічені парами (умова-оператор) [12].

$$P = < S, E, s_0 S^*, \delta >,\ \delta : E \to (U, Y)$$

Вершини графа P – суть контрольні точки програми. Перехід від однієї точки до іншої супроводжується виконанням оператора y, якщо умова u у поточному стані пам'яті програми приймає значення *True*.

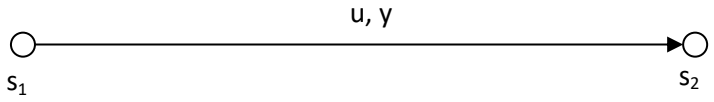


Рис. 1.1. Перехід в $U-Y$ програмі.

Виконання програми P починається в стані s_0 здійснюється недетермінованим чином й завершується в одному із заключних станів S^*.

Алгебраїчні моделі:

Алгебраїчні моделі імперативних структурованих програм використовують представлення програми термом (формулою) спеціальної програмної алгебри, операціями якої є оператори управління: послідовне виконання, розгалудження, повторення.

Наприклад:

1) Маємо дві програми A та B. Послідовне виконання програм позначаємо – $A; B$;

2) Розгалудження програм A, B з умовою q – $A_q^{\vee}B$;

3) Маємо програму A та постумову q. Повторення з постумовою – $\{A\}q$

4) Маємо програму A та передумову q. Повторення з передумовою - $q\{A\}$.

Таким чином, для задач аналізу програм та запису їх алгебраїчної моделі з тексту програми необхідно виділити перелік її вхідних параметрів, пам'ять рограми, перелік умов та операторів присвоювання.

Нехай X $=< x_1, \dots, x_n >$ - вектор змінних, які ми інтерпретуємо як змінні програми. A – алгебра з сигнатурою $\sum =< \sigma_1, \dots, \sigma_k >$ та носієм A. Визначимо X як пам'ять програми, а вектори a $=< a_1, \dots, a_n >$, $a_i \in A$ як стани памяті. Множина $U = A^n$ - універсум станів памяті програми.

Обчислюючим оператором є відображення $F : U \to U$, визначене покоординатно системою присвоювань

$$x_1 := f_1(\boldsymbol{X}), \dots, x_n := f_n(\boldsymbol{X}) \qquad (1.1)$$

Запишемо обчислюючий оператор у векторній формі **X := F(X)**, а вектор координатних функцій (обчислень) – у формі **F** = <f_1, …, f_n>.

Відзначимо, що в таких позначеннях обчислюючий оператор інтерпретується як одночасне присвоєння вектору змінних значень - правих частин покоординатних операторів присвоєнь.

Поняття лінійно визначеної програми:

Програма P визначена лінійно (інтерпретована над векторним простором), якщо ***A*** – деяке поле (область цілосності), а всі її обчислюючі оператори записано у наступній формі:

$$\mathbf{X := F*X + b}, \tag{1.2}$$

де **F** – лінійний оператор, який діє у векторному просторі **U,** а $\mathbf{b \in U}$.

Таким чином, програми, інтерпретовані над лінійними просторами, на кожному кроці обчислень виконують лінійні перетворення універсуму станів памяті.

1.2. Статичний аналіз програм. Інструменти статичного аналізу

Більшість методів аналізу програмного забезпечення можна розділити на методи статичного та динамічного аналізу. Статичний аналіз, на відміну від динамічного, не потребує виконання програми та повної реалізації програмного коду, але, у свою чергу, потребує доступу до всіх вихідних текстів програмної системи, включаючи об’єктні та завантажувальні модулі, інформацію, пов’язану з середовищем компіляції та виконанням програмних компонентів [13–15]. Таким чином, перевагою статичного аналізу є можливість його застосування на ранніх етапах розробки ПЗ [16], що, у свою чергу, є економічно-ефективним, оскільки дозволяє значно вплинути на якість, надійність та безпеку вихідного коду. А, відповідно, і зменшити ресурсні витрати на етапі тестування та імплементації програмного засобу.

Багато всесвітньовідомих компаній, незважаючи на деякі недоліки статичного аналізу (можливе виявлення не існуючих дефектів та помилок, більші часові затрати, тощо), визначають його обов’язковим етапом розробки програмного забезпечення, зокрема критичного. Необхідність застосування

інструментів статичного аналізу визначено у вимогах NASA [17] та стандартах ESA [18]. Необхідність використання статичного аналізу відзначено такми промисловими галузями, як медична (розробка медичних пристроїв та систем), авіаційна, ядерна (системи управління та захисту реакторів), тощо.

Методи статичного аналізу артефактів життєвого циклу можна розділити на два види: контроль того, що все формалізовані правила коректності побудови цих артефактів виконано, і пошук типових помилок і дефектів в них на основі деяких шаблонів. Часто інструменти статичного аналізу використовують обидва типи перевірок.

На даний час є три покоління інструментів статичного аналізу.

Інструменти статичного аналізу першого покоління використовувалися для пошуку найпростіших програмних помилок. В основі таких інструментів лежить пошук по сигнатурам, а саме пошук співпадінь з сигнатурами, включеними до бази перевірок. Одним із перших аналізаторів є **Lint** [19].

Інструменти статичного аналізу другого покоління зявилися на початку 2000-х років [20]. Їх основною особливістю є перехід від пошуку «підозрілих» фрагментів коду до аналізу дефектів часу виконання. Для цього було поєднано міжпроцедурний аналіз та аналіз шляхів виконання для аналізу станів передачі управління від однієї функції до іншої в програмній системі [21]. Таким чином, з’явилася можливість будувати граф потоку управління та граф потоку даних, що, власне, представляють модель виконання програми та модель залежностей одних змінних від інших. На відміну від аналізаторів першого покоління, зявилася можливість пошуку більш складних дефектів та зменшилася кількість «хибнопозитивних» результатів [21, 22].

Розвиток алгоритмів та SMT розвязувачів зумовив появу інструментів статичного аналізу третього покоління [23], в основу яких покладено функціональні можливості попередників в поєднанні з застосуванням розвязувачів. У своїй роботі [21] Чельф та Чу зазначають, що за даними

більш ніж десяти проєктів з відкритим вихідним кодом на http://www.scan.coverity.com, що є спільним підприємством Coverity, Symantec та Міністерства національної безпеки США, імплементація SAT розвязувачів до загального аналізу дозволила знизити кількість «хибнопозитивних» результатів на 15%.

На сьогодні розроблено велику кількість інструментів статичного аналізу, зокрема для різних мов програмування:

— для C/C++ - Cppcheck [24],Clang Static Analyzer [25],Clang-Tidy [26], Frama-C [27], Lint [28], Parasoft C/C++test [29], PC-Lint [30], Helix QAC [31] та ін.

— для **C# -** ReSharper [32], FxCop [33], Roslyn Analyzers [34], Security Code Scan [35], Roslynator [36], CodeRush [37], Parasoft dotTEST [38], та ін.

— для Java – FindBugs [39], SpotBugs [40], IntelliJ IDEA [41], SonarJava [42] , та ін..

— для Python – PyCharm [43], PyDev [44], Pylint [45] , та ін..

Деякі з розроблених аналізаторів підтримують декілька мов програмування – Coverity [46], Klocwork Insight [47], Checkmarx CxSuite [48].

Крім того, популярності набувають інструменти статичного аналізу, побудовані на базі машинного навчання (Machine Learning (ML)) – DeepCode [49], SapFix [50], Source{d} [51], CodeGuru [52], Infer [53].

Розглянемо більш детально деякі з основних аналізаторів:

Lint. Статичний аналізатор коду для мови Сі першого покоління, один із перших аналізаторів. Дав назву багатьом сучасним аналізаторам (cpplint, Puppet Lint, Rpmlint та ін.) [28].

Cppcheck - інструмент статичного аналізу коду C / C ++. Забезпечує унікальний аналіз коду для виявлення помилок і фокусується на виявленні невизначеної поведінки та небезпечних конструкцій коду. Основною метою розробників є зменшення кількості помилкових спрацьовувань. Cppcheck має можливість аналізувати код з нестандартним синтаксисом (вбудовані проєкти

та інші проєкти, які використовують різні мовні розширення). Доступні як інтерфейс командного рядка, так і графічний користувальницький інтерфейс. Інтегрується з багатьма інструментами розробки. Не підтримує всі конструкції, описані новими стандартами мови C ++. Безкоштовною є лише діюча версія інструменту, без можливості безкоштовного доопрацювання або доповнення функціоналу [24].

Clang Static Analyzer. Статичний аналізатор коду для мов C / C ++ / Objective-C, вбудований в компілятор Clang. Можна запускати або з командного рядка, або в Xcode для macOS. Аналізатор повністю відкритий і є частиною проєкт у Clang. Як і інші частини Clang, аналізатор реалізовано у вигляді бібліотеки C ++, яка може використовуватися іншими інструментами і додатками. Є можливість дописувати власний функціонал або робити запити розробникам. Розробники зазначають необхідність підвищення точності і обсягу алгоритмів аналізу, а також видів помилок, які має виявляти аналізатор. Зважаючи на те, що деякі алгоритми, необхідні для пошуку помилок, вимагають експоненціального часу, робота інструменту не є швидкою [25].

Clang-Tidy - інструмент «Лінтера» C ++ на основі Clang. Його мета - надати розширювану структуру для діагностики та виправлення типових помилок програмування, таких як порушення стилю, неправильне використання інтерфейсу або помилки, які можуть бути виявлені за допомогою статичного аналізу. Clang-tidy є модульним і надає зручний інтерфейс для написання нових перевірок. [26].

Frama-C. Аналізатор програм мовою C з відкритим вихідним кодом. Дозволяє перевірити відповідність вихідного коду наданій формальній специфікації. Для написання функціональних специфікацій можна використовувати спеціальну мову - ACSL. Специфікації можуть бути частковими, концентруючись на одному аспекті аналізованої програми за раз. Frama-C дозволяє спостерігати набори можливих значень змінних програми в кожній точці виконання; нарізати вихідну програму на спрощені;

переміщатися по потоку даних програми від визначення до використання або від використання до визначення [27].

Parasoft C/C++test. Повністю інтегроване рішення для тестування програмного забезпечення для вбудованих систем, критичних до безпеки. Його можливості автоматичного тестування програмного забезпечення також створені для сучасних високошвидкісних середовищ Agile DevOps. Він тісно інтегрується в середовище розробки C і C ++, конвеєр CI / CD і контейнерні розгортання для більш раннього виявлення дефектів і автоматичного забезпечення відповідності галузевим стандартам. У Parasoft C / C ++ test використовується найповніший набір методів статичного аналізу коду C (аналіз на основі шаблонів, аналіз потоку даних, абстрактна інтерпретація, метрики і т. д.), Перевірка якості коду за допомогою найбільшої кількості засобів перевірки в галузі, забезпечує дієві робочі процеси, які допоможуть команді розставити пріоритети в висновках і виправити дефекти в коді. Статичний аналізатор Parasoft C / C ++ test забезпечує найбільш повне покриття загальних стандартів безпеки, стандартів функціональної безпеки та інших галузевих стандартів кодування [29].

PC-lint Plus. Пропонує широкий спектр можливостей для аналізу якості програмного забезпечення. Дозволяє виявляти проблеми безпеки вказівника та часу життя пам'яті, такі як переповнення буфера та використання після звільнення, за допомогою механізму аналізу потоку даних відстеження значень і вбудованої семантики функцій для стандартних бібліотек C і C. Перевіряє ПЗ на наявність порушень внутрішніх правил коду, а також його відповідність галузевим стандартам, таким як MISRA®, AUTOSAR® і CERT® C. Є можливість налаштування цілої низки повідомлень для підтримки безлічі різних рекомендацій з написання коду; наприклад, використання фігурних дужок для керуючих структур, присвоювання в умовних виразах, явне уточнення пріоритету операторів і багато іншого. Точний набір бажаних повідомлень може бути вказаний в повторно використовуваних файлах

конфігурації для обміну між проєктами та командами. Безкоштовною є лише 30-денна пробна версія [30].

Helix QAC. Статичний аналізатор для мов C, C ++. Сертифікований на відповідність функціональної безпеки SGS-TÜV, включаючи IEC 61508, ISO 26262, EN 50128, IEC 60880 та IEC 62304, а також ISO 9001. Є можливість налаштування для підтримки стандартів кодування користувача. Helix QAC будує точну модель поведінки ПЗ та відслідковує значення змінних в коді, відповідно до певних станів памяті (по аналогії із запущеним додатком). Таким чином, аналізатор запезпечує максимальне покриття коду та зводить до мінімуму помилкові спрацювання та заперечення. Розпізнає проблеми, викликані занадто складним кодом. Підтримує більшість компіляторів та може інтегруватися з багатьма інструментами розробки, включаючи IDES (наприклад, Microsoft Visual Studio), системи контролю версій (наприклад, Helix Core) та сервери збірки неперервної інтеграції (наприклад, Jenkins). Можна запросити пробну версію [31].

ReSharper. Не є статичним аналізатором в класичному розумінні, оскільки надає мало сценаріїв використання. Розширює Visual Studio більш ніж 2200 перевірками коду на льоту для C #, VB.NET, ASP.NET, JavaScript, TypeScript і інших технологій. Для більшості перевірок ReSharper надає швидкі виправлення для поліпшення коду. Безкоштовною є лише пробна версія [32].

FxCop. Безкоштовний інструмент для статичного аналізу коду від компанії Microsoft. Виконує аналіз байт-коду (CIL) на відповідність рекомендаціям Microsoft з проєктування додатків. На даний момент проєкт не підтримується [33].

Roslyn Analyzers (.NET Compiler Platform). Набір статичних аналізаторів коду для мов C # і Visual Basic. Виконують аналіз вихідного коду на рахунок стилю, якості та зручності обслуговування, дизайну та інших проблем. Деякі з аналізаторів убудовано в Visual Studio. Більшість аналізаторів перевіряють саме стиль коду, і лише декілька – якість. У складі

цього проєкту також було проведено імпорт найбільш важливих правил FxCop [34].

Security Code Scan. Статичний аналізатор коду на основі .Net Compiler Platform для мов C # і Visual Basic. Виконує пошук різноманітних патернів помилок: впровадження SQL, міжсайтовий скриптинг (XSS), підробка міжсайтових запитів (CSRF), впровадження зовнішніх XML-об'єктів (XXE) та ін. Аналізує проєкти .NET и .NET Core у фоновому режимі або під час збірки. Працює з Visual Studio 2015 та її новішими версіями [35].

CodeRush. Плагін для Visual Studio. Продукт комерційний, але є пробна версія. Функція діагностики аналізу коду допомагає виявляти і виправляти можливі проблеми коду. Можна використовувати Вікно проблем коду, щоб запустити CodeRush для аналізаторів Roslyn в активному додатку або відобразити проблеми коду Visual Studio. CodeRush включає в себе функції швидкої навігації та швидкої навігації по файлам, які дозволяють швидко і легко знаходити символи і відкриті файли [37].

Parasoft dotTEST. Набір інструментів для тестування додатків .NET, що включає в себе статичний аналізатор коду. Виконує перевірку коду на відповідність галузевим стандартам безпеки, включаючи аналіз вимог та документації, необхідних для перевірки відповідності; виявлення проблем часу виконання (наприклад, нульові вказівники, втрати пам’яті, і т.ін.); пошук дублікатів коду; аналіз складності та структури коду. Можна використовувати як у Visual Studio, так і з використанням інтерфейсу командного рядка для сценаріїв автоматизації та непервної інтеграції. Продукт комерційний, є пробна версія [38].

FindBugs. Безкоштовний статичний аналізатор коду для Java. Аналізує байт-код програми. Шукає екземпляри «шаблонів помилок» - екземпляри коду, які можуть бути помилками. Працює на платформах GNU / Linux, Windows і MacOS X. Проєкт не підтримується на даний час [39].

SpotBugs. Програма, яка використовує статичний аналіз для пошуку помилок в коді Java. Це безкоштовне програмне забезпечення, яке

розповсюджується на умовах Стандартної громадської ліцензії обмеженого застосування GNU. SpotBugs перевіряє понад 400 шаблонів помилок. Є продовженням проєкту FindBugs [40].

SonarSource. Статичний аналізатор коду для Java, що розробляється компанією SonarSource. Використовує такі методи аналізу коду, як зіставлення зі зразком та аналіз потоку даних для пошуку "запахів" коду, помилок і вразливостей безпеки. Використання аналізатору доступне в Eclipse, IntelliJ і VSCode для розробників (SonarLint), а також у всьому ланцюжку розробки для автоматичного аналізу коду за допомогою автономного SonarQube або хмарного SonarCloud [42].

Coverity. Вважається одним із кращих статичних аналізаторів коду. Виконує статичний аналіз для C , C ++ , C # , Java , JavaScript , PHP , Python , .Net Core , ASP.NET , Objective-C , Go , JSP, Ruby , Swift , Fortran , Scala , VB.NET , ioS та Typescript. Підтримує більше 70 різних фреймворків для Java, JavaScript, C # та інших мов [46]. Підтримує програму з безкоштовної перевірки відкритих додатків.

Klocwork Insight. Один з лідерів ринку статичних аналізаторів коду. Статичне тестування безпеки додатків (SAST) Klocwork для C, C ++, C # і Java виявляє проблеми безпеки, якості та надійності програмного забезпечення, допомагаючи забезпечити дотримання стандартів. Створений для корпоративного DevOps, Klocwork масштабується для проєкт ів будь-якого розміру, інтегрується з великими складними середовищами і широким спектром інструментів розробника, а також забезпечує контроль, спільну роботу і звітність. Механізм диференціального аналізу забезпечує швидкі результати, зберігаючи при цьому точність, і легко інтегрується з конвеєрами CI/CD для автоматизації постійної відповідності - захищаючи програмне забезпечення від вразливостей при кожній фіксації [47].

Checkmarx CxSuite. Гнучкий додаток для статичного аналізу корпоративного рівня. Виявляє сотні вразливостей і слабких місць в користувальницькому коді; підтримує більше 22 мов (Java, C#, Visual Basic,

C/C++, Ruby, JavaScript, Perl та ін.) і середовищ програмування та сценаріїв з нульовою конфігурацією, необхідною для сканування будь-якої мови [48].

DeepCode. Виконує семантичний аналіз коду. Може використовуватися в режимі реального часу як розширення середовища розробника IDE або безпосередньо в робочому процесі Git. Оскільки основною ідеологією аналізатора є машинне навчання та штучний інтелект, замість зарезервованого експертами набору правил, DeepCode аналізує коміти з відкритим вихідним кодом та оновлює базу існуючих помилок. Використовує Datalog солвер [49]. Використання штучного інтелекту дозволяє не лише показати користувачу, що є логічно неправильним у коді, а і показати можливі варіанти вирішення проблеми.

Infer. Інструмент статичного аналізу для C/C++, Objective-C та Java, від Facebook. Перевіряє виключення нульового вказівника, втрату ресурсів, доступність анотацій, відсутність засобів захисту блокувань та умови конкуренції в коді Android та Java, і, відповідно, розіменування нульового вказівника, втрату памяті, угоди про кодування та недоступні API для C, C ++ та iOS / Objective-C. Продовжується робота над удосконаленням аналізатора [53].

Sapienz та SapFix. Sapienz - інтелектуальний автоматичний інструмент тестування ПЗ, розроблений Facebook. Використовує інструмент статичного аналізу Infer [53]. SapFix - новий гібридний інструмент для автоматичного пошуку та виправлення помилок ПЗ. Аналогічно DeepCode має можливість автоматичного створення виправлень для певних видів помилок. На даному етапі SapFix орієнтовано на виправлення помилок, знайдених Sapienz [54]. Можуть працювати як разом так і окремо. На даний час інструменти знаходяться на стадії розробки та тестування. Використовуються для перевірки коду Facebook, Instagram, Workplace та Messenger.

CodeGuru. Використовує аналіз програм та машинне навчання для визначення потенційних дефектів ПЗ та пропонує можливі варіанти покращення кода для Java. Не показує синтаксичні помилки. Працює з такми

напрямами аналізу як – кращі практики AWS (Amazon Web Services), паралелізм, попередження втрати ресурсів та конфіденційної інформації, загальні передові практики кодування, рефакторінг, перевірки введення, тощо [52].

Найбільш складні види аналізу [55] виконують PolySpace Verifier [56], Coverity Prevent [57] та Klocwork K7 [58]. Перевірені на практиці правила коректності коду або шаблони типових помилок переносяться в середовища розробки, такі як Eclipse або Microsoft Visual Studio, і поступово стають семантичними правилами мов програмування, їх перевірка покладається вже на компілятори цих мов. Тому статичний аналіз можна вважати найбільш широко застосовуваним методом верифікації.

Якщо в проєкті використовуються мови опису архітектури або графічні мови проєктування, побудовані з їх допомогою артефакти можна також перевіряти за допомогою спеціалізованих інструментів [59].

1.3. Методи та техніки статичного аналізу програм

Серед існуючих методів та технік статичного аналізу програм можна виділити наступні:

1. ***Counter Example Guided Abstraction Refinement (CEGAR)*** [60] - метод статичного аналізу коду, що базується на абстракції та уточнені на основі контрприкладів. Для перевірки властивостей програми за допомогою CEGAR постає необхідність у формулюванні їх у вигляді недосяжності заданої множини помилкових станів у програмі. Відповідно, абстрактна модель програми буде уточнюватися доти, доки не буде доведено її коректність, або не буде знайдено помилковий шлях (шлях, який починається в точці входу і веде до помилкового стану). Підхід CEGAR складається з трьох етапів – побудови та перевірки абстракції, перевірки виконуваності та уточнення абстракції. Так, на першому етапі, відповідно заданих правил відповідності між абстрактними станами та переходами в програмі, будується абстракція програми з усіма помилковими шляхами виконання.

Після чого, виконується перевірка побудованої абстракції на наявність помилок. В залежності від процесу перевірки (після побудови абстракції або протягом етапу побудови), можуть використовуватися і сторонні інструменти перевірки моделей. Зокрема, є можливим застосування методів лінивої абстракції, предикатної абстракції, методу адаптивного аналізу (CPA), аналізу з явними значеннями. Виконання першого єтапу закінчується або знаходженням помилкового шляху (контрприкладу), або доказом відсутності помилок моделі. Відсутність помилок в абстрактній моделі говорить і про відсутність помилок в вихідній програмі. При наявності контрприкляду, останній передається на перевірку його виконуваності в вихідній програмі (за рахунок побудови формули шляху, кодуючої виконання програми на шляху до помилки та передачі її вирішувачу). Якщо шлях невиконуваний, постає необхідність в уточнені абстрації програми.

2. ***Предикатна абстракція*** [61, 62]. Станами моделі є набори предикатів, які описують множини конкретних станів програми. Зазвичай будується абстракція існування (existential abstraction), в якій існування переходу між двома конкретними станами вказує на існування переходу між будь-якими двома абстрактиними станами, які включають дані конкретні стани. Existential abstraction гарантує збереження досяжних шляхів абстрактної моделі в помилковий стан, і відповідно, визначає відсутність помилки у програмі при її відсутності в абстрактній моделі. Предикати станів можуть бути виражені в логіці висловлень, логіці першого порядку або логіці більш високих порядків. Функціональні символи та предикати можуть мати додаткову інтерпретацію у вигляді теорій цілих та раціональних чисел, списків, масивів, бітових векторів, тощо. Обчислення переходів між станами потребує представлення конструкцій мови програмування у вигляді логічних формул з використанням різних логік та теорій.

Найбільш поширеними варіантами предикатних абстракцій є декартова та булева предикатні абстракції. Для декартової абстракції обчислення стану зводиться до превірки істинності кожного окремого предикату. А, власне

самою абстракцією, є найбільш строга коньюнкція предикатів, які покривають стани програми (предикати можуть поєднуватися лише коньюнкцією). Обчислення булевої предикатної абстракції потребує отримання набору векторів істинних предикатів, а абстракцією є найбільш строга формула, яка покриває стани програми (допускаються будь-які комбінації предикатів у логіці висловлювань).

3. ***Абстрактне дерево досяжності (ART - Abstract Reachability Tree)*** [63-65] – містить шляхи, досяжні з початкового стану. Вершини дерева – стани програми (вміщують, як приклад, вершину графу потоку керування та стан предикатної абстракції). Ребра – переходи, що складаються з однієї або декількох інструкції графу потоку керування. Побудова дерева полягає в обчисленні кінцевого стану по стану в поточній вершині та переходу, яким помічено ребро. Побудова дерева досяжності виконується одночасно з перевіркою заданої властивості програми. У випадку перевірки властивості досяжності, побудова дерева зупиняється у випадку виявлення помилкового стану. Скінченність дерева досяжності забезпечується скінченністю множини абстрактних станів. Крім того, не перебираються шляхи, які покриваються іншими абстрактними станами. Покриття станів означає вкладеність відповідних множин станів програми.

4. ***Лінива абстракція*** – дозволяє не перебудовувати абстрактне дерево досяжності повністю при уточненні абстрактного домену. Після аналізу контрприкладa виконується частковий перегляд дерева – лише для тих шляхів, в яких його перегляд може вплинути на наявність помилкових станів. Знаходиться перша з кінця вершина, у якій помилка є недосяжною та виконується перегляд шляхів, які починаються саме з цієї вершини. Алгоритми лінивої абстракції та деякі пропозиції щодо їх удосконалення розглянуто в [66, 67].

5. ***Адаптивний статичний аналіз***. Головною ідеєю є обєднання аналізу потоків даних та методів перевірки моделей, і, відповідно, розробка підходу конфігурації їх взаємозвязку. Конфігурація в аналізі потоку даних

полягає у виборі відповідного інтерпретатора (абстракної множини, оператора розширення, функції переходу). Конфігурацію декількох інтерпретаторів розглянуто в [68-70]. Таким чином, метою адаптивного статичного аналізу є створення методу конфігуруємого аналізу на основі використання готового набору існуючих реалізацій алгоритмів аналізу. Для комбінації декількох алгоритмів аналізу одночасно, визначаються складовий оператор злиття та складовий оператор зупинки, які, в свою чергу, складено із компонент, які є відповідними операторами злиття та зупинки алгоритмів методу адаптивного аналізу (CPA) [71-72].

Використання адаптивного статичного аналізу в деяких випадках дозволяє збільшити швидкість аналізу та може впливати на точність.

6. ***Аналіз з явними значеннями***. Для кожної змінної, у даному виді аналізу, абстрактним станом є множина усіх її можливих значень. Перевищення деякого встановленого порогу значень говорить про те, що змінна може приймати будь-які значення [72].

Конструюючи переходи необхідно враховувати семантику інструкцій, присвоюючих змінним із абстрактного стану явні значення, константи або вирази, які можуть бути обчислені в константи у даному абстрактному стані, а також, семантику виразів в умовних операторах, для яких можна вивести явне значення змінної.

Даний вид аналізу є достатньо ефективним при невеликій кількості змінних та їх можливих значень. У противному випадку, він втрачає точність або потребує великої кількості ресурсів.

7. ***SMT/SAT вирішувачі або вирішуючі процедури*** [73, 74] – алгоритми, які приймають на вхід задачу розвязуваності (питання, сформульоване в межах формальної системи) та видають відповідний конкретний результат («так» або «ні»); розглядаються для алгоритмічно розвязуваних задач, сформульованих в межах теорій першого порядку.

Відповідно до підтримуваних логік та теорій можуть бути розподілені за декількома типами:

- SAT вирішувачі – вирішувачі для логіки висловлювань; інструменти для розвязання задач виконувасті булевих формул.

- SMT вирішувачі – вирішувачі для формул класичної логіки першого порядку з рівністю, заданих в комбінації деяких аксіоматичних теорій.

Найбільш використовуваними на практиці є такі теорії як раціональна та цілочисельна лінійна арифметики, неінтерпретовані функції, масиви, бітові вектори.

8. ***Інтерполяція*** – використовується на етапі уточнення абстрактного домену для уточнення абстракції, і, в першу чергу, - для уточнення предикатів в предикатній абстракції. Метод інтерполяції полягає у використанні інтерполянтів Крейга дозволяючих знаходити неявні залежності [67, 75].

Використання методів інтерполяції та предикатів дозволяє визначати точки програми, для яких відповідні предикати є корисними, що, в свою чергу, дозволяє розглядати різні множини предикатів для різних точток програми, і, відповідно, перебудовувати абстракцію лише тих частин програми, які повязані з наборами предикатів, що змінилися.

9. ***Метод обмежувальної перевірки моделей (BMC, Bounded Model Checking)*** [76] - базується на розгортанні графу станів програми на скінченну кількість кроків. На кожному кроці виконується перевірка порушення специфікації не більше ніж за поточну обрану кількість кроків. Якщо порушення не виявлено, виконується перевірка умов розгортання для виключення подальшого розгортання графу по нездійсненним шляхам. Оскільки ні одна із умов розгортання не виконується, коректність вихідної програми доведено. В іншому випадку, кількість кроків збільшується і перевірка повторюється до моменту виявлення порушення специфікації, або виходу за межі максимальної кількості кроків.

10. ***Аналіз рекурсивних структур даних*** [77, 78] – аналіз, який моделює потенційно нескінченні рекурсивні структури даних в кучі, такі як списки, множини, відображення, дерева і т.д., використовуючи абстракції,

основані на графах звязків. Класи звязків, які задаються для побудови графів, описують звязки між елементами структури даних, які необхідно відслідковувати, задають предикати на дані, які зберігаються в елементах. Метод дозволяє використовувати предикати, які тяжко знаходити іншими методами.

11. ***Модульний аналіз*** – можливість аналізу програми по частинам. Найбільш швидкі підходи окремо аналізують кожну функцію програми, створюючи відповідні анотації, які зберігають значимі результати аналізу для кожної функції. Крім того, забезпечення модульності є можливим не лише на рівні функцій. Так, в абстрактному дереві досяжностей запамятовуються результати аналізу ребер, які можуть вміщувати як виклики функцій повністю, так і окремі ітерації циклів [79]. Таким чином, прохід по аналогічним ребрам, дає можливість використання результатів попереднього аналізу, що значно впливає на швидкість аналізу програми.

12. ***Тестування*** – методи, які дозволяють генерувати вхідні параметри програми таким чином, щоб її виконання забезпечувало покриття тієї чи іншої її частини. Спрямування інструмента генерації тестів на покриття помилкового стану програми дає можливість використовувати їх для перевірки властивостей досяжності.

Окремо, вважаємо за необхідне, звернути увагу на ***метод програмних інваріантів,*** який полягає у перевірці інваріантності визначеної специфікаціями властивості програми. Особливість методу полягає, зокрема, у можливості його застосування як для однопоточних задач, так і для задач керування асинхронними потоками обчислень, на відміну від традиційних логічних методів та методів model checking.

Інструменти статичної верифікації мають відповідати наступним вимогам:

- Підтримка конструкцій програми, яка аналізується (булеві вирази, лінійні вирази, бітові вирази, вирази з вказівниками, рекурсивні структури даних, підтримка багатопоточності, тощо).

- Масштабованість (можливість застосування інструмента верифікації до великих розмірів вихідного коду програми).

- Доведення властивостей (інструменти, виконуючі доведення заданих властивостей, мають надавати гарантії того, що властивості дійсно не порушено).

- Властивості, що перевіряються – властивості досяжності, властивість завершуваності.

Реалізацію та використання описаних вище методів та технік статичного аналізу в деяких інструментах статичного аналізу представлено в Таблиці 1.1.

Таблиця 1.1.

Методи та техніки статичного аналізу в інструментах статичного аналізу

	Blast	**CPAchecker**	**Slam**	**FShell**	**Satabs**	**Wolverine**	**Llbmc**	**Yogi**	**Cbmc**
Counter Example Guided Abstraction Refinement (CEGAR)	Так	Так	Так		Так	Так		Так	
Предикатна абстракція	Так	Так	Так		Так			Так	
Абстрактне дерево досяжності (ART)	Так	Так				Так			
Лінива абстракція	Так	Так				Так			
Адаптивний статичний аналіз	Частково	Так							
Аналіз з явними значеннями	Так	Так							
Вирішуючі процедури (SMT/SAT вирішувачі)	SMT	SMT	SMT	SAT	SAT	SAT	SMT	SMT	SAT
Інтерполяція	Так	Так							
Метод обмежувальної перевірки моделей (BMC)				Так			Так		Так
Аналіз рекурсивних структур даних									
Модульний аналіз		Частково						Так	
Тестування				Так				Так	

Висновки до розділу 1

Проблема статичного аналізу програмного забезпечення, зокрема, проблема верифікації, є однією з найважливіших проблем при розробці надійного та ефективного програмного забезпечення обчислювальних систем. Не зважаючи на те, що у загальному вигляді ця проблема є алгоритмічно нерозв’язуваною, її ефективне розв’язання можливе для широкого класу спеціалізованих предметних областей.

Методи верифікації за допомогою статичного аналізу або вже пройшли апробацію на практиці і використовуються в комерційних інструментах і широко застосовуваних інструментах розробки загального призначення, або все ще залишаються в ранзі новаторських, дослідницьких робіт. Дослідницькі методи на даний момент, в основному, пов'язані з формалізацією різних характеристик і властивостей програмного забезпечення.

Основними підходами статичного аналізу програм, покладеними в основу реалізації сучасних інструментів статичного аналізу, є Counter Example Guided Abstraction Refinement (CEGAR) та Метод обмежувальної перевірки моделей (BMC).

Ефективність та точність роботи інструментів статичного аналізу, в більшій мірі, визначається можливостями використовуваних вирішувачів. Збільшення точності моделювання семантики мови, потребує більшої кількості ресурсів для аналізу, але, у свою чергу, гарантує меншу кількість хибних повідомлень про помилки. Використання різних підходів та методів зумовлює надання різними інструментами верифікації різних гарантій відсутності помилок. Так, наприклад, інструменти на основі CEGAR показують відсутність помилок з точністю до підтримуваних конструкцій мови C, а інструменти на основі BMC лише перевіряють відсутність помилок до заданого у якості параметру обмеження на кількість ітерацій циклів.

Поєднання різних методів, що дозволяють збільшити точність і швидкість верифікації на сьогодні залишається найперспективнішим напрямом досліджень. Зважаючи на залежність функціональних можливостей сучасних статичних аналізаторів від функціональності використовуваних ними вирішувачів, останні залишаються об'єктом багатьох активних досліджень у всьому світі, як у науковому середовищі, так і в промисловості. Відповідно, особливо актуальною залишається і розробка спеціальних алгоритмів статичного аналізу.

У даному розділі:

1. представлено огляд існуючих спеціалізованих систем статичного аналізу програм;
2. розглянуто методи та техніки статичного аналізу програм та їх використання в деяких інструментах статичного аналізу;
3. приведено базисні дефініції дисертаційного дослідження – визначено загальні поняття графових та алгебраїчних моделей програм, приведено визначення поняття лінійно визначених програм.

Список літератури до Розділу 1

1. Ершов А.П. Введение в теоретическое программирование: беседы о методе. / А.П. Ершов. – М.: Наука, 1977. – 288 с.

2. Основи дискретної математики / [Ю. Капітонова, С. Кривий, О. Летичевський та ін.]. – Київ: Наукова думка, 2002. – 580 с.

3. Львов М. С. О реализации вычислений в задачах анализа программ, определенных над векторными пространствами / М. С. Львов. // Проблемы программирования. – 2004. – №2. – С. 95–101.

4. Львов С. М. О технологиях построения и обработки математических моделей программ / С. М. Львов // Проблемы программирования. – 2007. - №3. – С. 41-48.

5. Львов С. М. Инвариантные равенства малых степеней в программах, определенных над полем / С. М. Львов // Кибернетика. – 1988. - №1 – С. 108-110.

6. Letichevsky A. Discovery of invariant equalities in programs over data fields. / A. Letichevsky, M. Lvov. // Applicable Algebra in Engineering, Communication and Computing. – 1993. – №4. – С. 269–286.

7. Lvov M. S. About one algorithm of program polynomial invariants generation. / M. S. Lvov // Workshop on Invariant Generation, WING 2007 : Hagenberg, Austria: June 25–26. - 2007. – С. 85-99.

8. Львов М. С. Статический анализ физических размерностей переменных программ и его реализация в алгебраическом программировании / М. С. Львов. // Проблеми програмування. – 2015. - №2. – С. 3-12

9. Iterative methods of program analysis / A.Godlevskii, Y. Kapitonova, S. Krivoi, A. Letichevskii. // Cybernetics. – 1989. – №25(2). – С. 139–152.

10. Müller-Olm M., Seidl H. Computing polynomial program invariants / M. Müller-Olm, H. Seidl // Information Processing Letters. - 2004. - 91(5). – С. 233-244.

11. Львов М. С. Основи комп’ютерної алгебри та алгебраїчних обчислень : навч. посіб. / М. С. Львов. - Херсон : ТОВ «ВКФ «СТАР» ЛТД», 2018. - 238 с.

12. Летичевский А. А. Об одном подходе к анализу программ / А. А. Летичевский // *Кибернетика*. – 1979. - №6. – С. 1 – 8.

13. The experience of comparison of static security code analyzers / A. Markov, A. Fadin, V. Shvets, V. Tsirlov // International Journal of Advanced Studies. - 2015. - №5(3). – С. 55-63.

14. Multilevel metamodel for heuristic search of vulnerabilities in the software source code. / A. Markov, A. Fadin, V. Shvets, V. Tsirlov // International Journal of Control Theory and Applications. – 2016. - №9(30). – С. 313-320.

15. Аветисян А. Технологии статического и динамического анализа уязвимостей программного обеспечения / А. Аветисян, А. Белеванцев, И. Чукляев. // Вопросы кибербезопасности. – 2014. – №3. – С. 20–28.

16. Lopes R. Static Analysis tools, a practical approach for safety-critical software verification / R. Lopes, D. Vicente, N. Silva // ESA Special Publication. - 2009 – 669 с.

17. Miller J. Design Development Test and Evaluation (DDT and E) Considerations for Safe and Reliable Human Rated Spacecraft Systems / J. Miller, J. Leggett, J. Kramer-White. - Volume II. – 2008. – 697 с.

18. ECSS-E-40 Part 2B: Software – Part 2: Document requirements definitions (DRDs) [Електронний ресурс]. – 2005. – Режим доступу до ресурсу: https://ecss.nl/standard/ecss-e-40-part-2b-software-part-2-document-requirements-definitions-drds/.

19. Lint [Електронний ресурс]. – Режим доступу до ресурсу: http://tack.sourceforge.net/olddocs/lint.pdf.

20. Emanuelsson P., Nilsson, U. A comparative study of industrial static analysis tools / P. Emanuelsson, U. Nilsson // Electronic notes in theoretical computer science. – 2008. - №217. – С. 5-21.

21. Checking system rules using system-specific, programmer-written compiler extensions / D. Engler, B. Chelf, A. Chou, S. Hallem // OSDI'00 Proceedings of the 4 th conference on Symposium on Operating System Design and Implementation. – 2000. -№4. – 16 с.

22. Why don't software developers use static analysis tools to find bugs? / B. Johnson, Y. Song, E. Murphy-Hill, R. Bowdidge // 35th International Conference on Software Engineering (ICSE). – 2013. – С. 672-681.

23. Franco J., Martin J. A History of Satisfiability / J. Franco, J. Martin// Handbook of satisfiability. – 2009. - №185, С. 3-74.

24. Cppcheck [Електронний ресурс]. – Режим доступу до ресурсу: http://cppcheck.sourceforge.net/

25. Clang Static Analyzer [Електронний ресурс]. – Режим доступу до ресурсу: https://clang-analyzer.llvm.org/

26. Clang-Tidy [Електронний ресурс]. – Режим доступу до ресурсу: https://clang.llvm.org/extra/clang-tidy/

27. Frama-C [Електронний ресурс]. – Режим доступу до ресурсу: http://frama-c.com/

28. Lint [Електронний ресурс]. – Режим доступу до ресурсу: http://tack.sourceforge.net/olddocs/lint.pdf - --- заменить

29. Parasoft C/C++test [Електронний ресурс]. – Режим доступу до ресурсу: https://www.parasoft.com/products/parasoft-c-ctest/c-c-static-analysis/

30. PC-Lint [Електронний ресурс]. – Режим доступу до ресурсу: https://gimpel.com/

31. Helix QAC [Електронний ресурс]. – Режим доступу до ресурсу: https://www.perforce.com/products/helix-qac

32. ReSharper [Електронний ресурс]. – Режим доступу до ресурсу: https://www.jetbrains.com/resharper/?gclid=EAIaIQobChMIoM-9gffs5wIVwQ8YCh0FOgw6EAAYASAAEgIzBPD_BwE

33. FxCop [Електронний ресурс]. – Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/visualstudio/code-quality/install-fxcop-analyzers?view=vs-2019

34. Roslyn Analyzers [Електронний ресурс]. – Режим доступу до ресурсу: https://docs.microsoft.com/en-US/visualstudio/code-quality/roslyn-analyzers-overview?view=vs-2019

35. Security Code Scan [Електронний ресурс]. – Режим доступу до ресурсу: https://security-code-scan.github.io/

36. Roslynator [Електронний ресурс]. – Режим доступу до ресурсу: https://marketplace.visualstudio.com/items?itemName=josefpihrt.Roslynator2019

37. CodeRush [Електронний ресурс]. – Режим доступу до ресурсу: https://www.devexpress.com/products/coderush/

38. Parasoft dotTEST [Електронний ресурс]. – Режим доступу до ресурсу: https://www.parasoft.com/products/parasoft-dottest/net-static-analysis/

39. . FindBugs [Електронний ресурс]. – Режим доступу до ресурсу: http://findbugs.sourceforge.net/

40. SpotBugs [Електронний ресурс]. – Режим доступу до ресурсу: https://spotbugs.github.io/

41. IntelliJ IDEA [Електронний ресурс]. – Режим доступу до ресурсу: https://www.jetbrains.com/idea/

42. SonarJava [Електронний ресурс]. – Режим доступу до ресурсу: https://www.sonarsource.com/java/

43. PyCharm [Електронний ресурс]. – Режим доступу до ресурсу: https://www.jetbrains.com/ru-ru/pycharm/

44. PyDev [Електронний ресурс]. – Режим доступу до ресурсу: https://www.pydev.org/

45. Pylint [Електронний ресурс]. – Режим доступу до ресурсу: https://pypi.org/project/pylint/

46. Coverity, "Coverity Static Analysis Data Sheet" (PDF). Synopsys.com. Retrieved 2019-07-15. [Електронний ресурс]. – Режим доступу

до ресурсу: https://www.synopsys.com/content/dam/synopsys/sig-assets/datasheets/SAST-Coverity-datasheet.pdf

47. Klocwork Insight [Електронний ресурс]. – Режим доступу до ресурсу: https://marketplace.visualstudio.com/items?itemName=AlenZukich.KlocworkInsight

48. Checkmarx CxSuite [Електронний ресурс]. – Режим доступу до ресурсу: https://www.checkmarx.com/

49. DeepCode [Електронний ресурс]. – Режим доступу до ресурсу: https://www.researchgate.net/publication/229131551_DATALOG_SOLVE_A_Datalog-Based_Demand-Driven_Program_Analyzer

50. SapFix [Електронний ресурс]. – Режим доступу до ресурсу: https://engineering.fb.com/2018/09/13/developer-tools/finding-and-fixing-software-bugs-automatically-with-sapfix-and-sapienz/

51. Source{d} [Електронний ресурс]. – Режим доступу до ресурсу: https://github.com/src-d

52. CodeGuru [Електронний ресурс]. – Режим доступу до ресурсу: https://docs.aws.amazon.com/codeguru/latest/reviewer-ug/welcome.html

53. Infer [Електронний ресурс]. – Режим доступу до ресурсу: https://fbinfer.com/, https://engineering.fb.com/developer-tools/sapienz-intelligent-automated-software-testing-at-scale/

54. Sapienz [Електронний ресурс]. – Режим доступу до ресурсу: https://engineering.fb.com/developer-tools/finding-and-fixing-software-bugs-automatically-with-sapfix-and-sapienz/

55. Emanuelsson P., Nilsson U. A comparative study of industrial static analysis tools / P. Emanuelsson, U. Nilsson // Electronic notes in theoretical computer science. - 2008. – №217. – С. 5-21.

56. PolySpace Verifier, Polyspace Static Analysis [Електронний ресурс]. – Режим доступу до ресурсу: http://www.mathworks.com/products/polyspace/index.html?s_cid=psr_prod.

57. Static Code Analysis [Електронний ресурс]. – Режим доступу до ресурсу: http://www.coverity.com/html/prod_prevent.html.

58. KlocworkEmbeddedSystems [Електронний ресурс]. – Режим доступу до ресурсу: http://www.iplbath.com/pdf/klocwork/Klocwork EmbeddedSystems.pdf.

59. A light-weight static approach to analyzing UML behavioral properties / Yu. Lijun, R. France, I. Ray, K. Lano // 12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007). - 2007, July. – IEEE. – C. 56-63.

60. Counterexample-guided abstraction refinement for symbolic model checking / [E. Clarke, O, Grumberg, S. Jha, at all.]. // *Journal of the ACM (JACM)*/ - 2003. - №*50*(5). – P. 752-794.

61. Graf S., Saidi H. Construction of abstract state graphs with PVS / S. Graf, H. Saidi // International Conference on Computer Aided Verification. Springer, Berlin, Heidelberg. – 1997. - P. 72-83.

62. Ball T., Rajamani S. The SLAM project: Debugging system software via static analysis / T. Ball, S. Rajamani // Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages. – 2002. - pp. 1-3.

63. The software model checker b last / D. Beyer, T. Henzinger, R. Jhala, R. Majumdar // International Journal on Software Tools for Technology Transfer. - 2007. - №9(5) – pp. 505-525.

64. Software model checking via large-block encoding / [D. Beyer, A. Cimatti, A. Griggio, at all.] // Formal Methods in Computer-Aided Design. – 2009. - pp. 25-32.

65. McMillan K. L. Lazy abstraction with interpolants / K. L. McMillan // International Conference on Computer Aided Verification. Springer, Berlin, Heidelberg. - 2006. - pp. 123-136.

66. Lazy abstraction / T. A. Henzinger, R. Jhala, R. Majumdar, G. Sutre, // Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages. - 2002. -pp. 58-70.

67. Abstractions from proofs / Henzinger, T. A., Jhala, R., Majumdar, R., & McMillan, K. L. // ACM SIGPLAN Notices. - 2004. - №39(1). – pp. 232-244.

68. Steffen B. Data flow analysis as model checking / B. Steffen // International Symposium on Theoretical Aspects of Computer Software. Springer, Berlin, Heidelberg. – 1991. - pp. 346-364.

69. Gulwani S., Tiwari A. Combining abstract interpreters / S. Gulwani, A. Tiwari // ACM SIGPLAN Notices. - 2006. - №41(6). - pp. 376-386.

70. Lerner S. Composing dataflow analyses and transformations / S. Lerner, D. Grove, C. Chambers // ACM SIGPLAN Notices. - 2002. - №37(1). - pp. 270-282.

71. Fischer J. Joining dataflow with predicates / J. Fischer, R. Jhala, R. Majumdar, // ACM SIGSOFT Software Engineering Notes. – 2005. - №30(5). - pp. 227-236.

72. Beyer D. Program analysis with dynamic precision adjustment / D. Beyer, T. Henzinger, G. Théoduloz // 23rd IEEE/ACM International Conference on Automated Software Engineering. – 2008. - pp. 29-38

73. Kroening D. Decision Procedures. An Algorithmic Point of View / D. Kroening, O. Strichman. – Berlin: Springer-Verlag Berlin Heidelberg, 2008. – 306 p.

74. Efficient satisfiability modulo theories via delayed theory combination / [M. Bozzano, R. Bruttomesso, A. Cimatti, at all.] // International Conference on Computer Aided Verification. Springer, Berlin, Heidelberg. - 2005. - pp. 335-349.

75. Schmidt D. A. Data flow analysis is model checking of abstract interpretations / D. A. Schmidt // Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages. – 1998. - pp. 38-48.

76. Jones N. D., Muchnick S. S. A flexible approach to interprocedural data flow analysis and programs with recursive data structures / N. D. Jones, S. S. Muchnick // Proceedings of the 9th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. - 1982. - pp. 66-74.

77. M. Sagiv Parametric shape analysis via 3-valued logic / M. Sagiv, T. Reps, R. Wilhelm // ACM Transactions on Programming Languages and Systems (TOPLAS). – 2002. - №24(3). – pp. 217-298.

78. Beyer D. Lazy shape analysis / D. Beyer, T. A. Henzinger, G. Théoduloz // International Conference on Computer Aided Verification. Springer, Berlin, Heidelberg. - 2006. - pp. 532-546.

79. Wonisch D. Block abstraction memoization for CPAchecker / D. Wonisch / International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer, Berlin, Heidelberg. - 2012. - pp. 531-533.

.

РОЗДІЛ 2. ГЕНЕРАЦІЯ ІНВАРІАНТІВ ПРОГРАМ

Успішне розв’язання задач аналізу, верифікації та оптимізації програм часто потребує знаходження інваріантних співвідношень в контрольних точках програми, тобто таких співвідношень між змінними програми, які виконуються (в даній контрольній точці) у процесі її роботи незалежно від початкового стану пам’яті. Однією з важливих проблем є проблема автоматичного генерування програмних інваріантів. Інваріанти програм використовуються, зокрема, в методах верифікації програм. Властивість коректності програми формулюється з точки зору її повної або часткової коректності. Доказ завершення програми має бути реалізовано окремо від доказу його часткової коректності. Таким чином, існує проблема пошуку інваріантів програми як ключової проблеми аналізу властивостей програм.

Поняття програмного інваріанту та проблему пошуку інваріантів циклів в імперативних програмах було представлено в роботах Р. Флойда [1] та С. Хоара [2] як ключова проблема процесу аналізу властивостей програм. Основу методу Хоара складають методи пошуку і генерації інваріантів програмних циклів з подальшим використанням дедуктивних методів доведення тверджень (теорем) про властивості програми. Цей метод є і на сьогоднішній день основним при верифікації програмних систем функціонального типу.

Використання інваріантів при верифікації зводиться до проблеми пошуку інваріантних співвідношень для заданої конструкції програми (наприклад, інваріантів циклу), а потім доведення, що з побудованої множини співвідношень слідує потрібний предикат (постумова), наявність якого гарантувало б правильність роботи програми. За методами розв’язання даної проблеми закріпилася назва «методи потокового аналізу програм». Виникнення цих методів пов'язане з практичною діяльністю в програмуванні і, перш за все, зі створенням якісного та надійного програмного забезпечення сучасних обчислювальних систем. Історично, першим методом потокового

аналізу вважається інтервальний метод, запропонований Коком, Шварцем [3] і розвинений згодом Аллен [4] та іншими авторами.

Дослідження задачі автоматичної генерації програмних інваріантів для різних алгебр даних виконувалися, починаючи в 70-х років, і в ІК НАН України. Так, на початку 1970 рр. Летичевським О.А. було визначено новий підхід до аналізу потоків даних [5].

Паралельно, відповідні дослідження було проведено Г. Кілдалом [6]. Кілдалл вивчив проблему пошуку в програмах інваріантних відношень вигляду r = 0, де r є змінною, а 0 - константою. В ітеративному методі Кілдалла інваріанти та їх послідовне наближення розглядаються як елементи напіврешіток, а пошук інваріантів для напростіших фрагментів програми - зводиться до обчислення значення функції (функція пошуку інваріантів). Ця особливість полягає у побудові деякої множини співвідношень по визначеному фрагменту (лінійний фрагмент) та множини співвідношень на його вході деякої множини співвідношень на його виході. Кілдалл детально вивчив випадок, коли функція пошуку інваріанта є дистрибутивною щодо основної операції напіврешітки і запропонував алгоритм та алгебраїчну інтерпретацію пошуку таких співвідношень. В [7] описано метод генерації нелінійного та, взагалі кажучи, неполіноміального інваріантного відношення для лінійних циклів. Метод використовує власні значення та власні вектори лінійного оператора в тілі циклу. Через п’ять років, детально вивчивши властивості алгоритму Кілдалла у випадку монотонної функції, Кам та Ульман [8] встановили нерозв'язність проблеми побудови повної системи інваріантів у загальному випадку, і запропонували деяке вдосконалення алгоритму Кілдалла.

У роботах як Кілдалла, так і Кама та Ульмана розглядалися задачі одностороннього аналізу потоку даних, тобто задачі, в яких властивості стану програми визначаються виключно властивостями її попередників (пряма задача потокового аналізу) або, навпаки, тільки властивостями її спадкоємців (зворотна задача потокового аналізу).

В семидесятих роках також вийшла низка праць, у яких Вегбрайт [9-11], Эльпас [12], Манна та Кац [13] запропонували записувати семантику операторів присвоювання в тілі циклу у вигляді рекурентних рівнянь, знаходячи явні вирази для значення кожної змінної як функції числа s ітерацій циклу і послідуюче виключення змінної s для отримання інваріантних предикатів. Але ефективність запропонованих методів для нетривіальних циклічних програм не було продемонстровано.

Основні результати досліджень ІК НАН України викладено також в [14, 15]. Так, в [14] А.А. Летичевський сформулював задачу опису множини інваріантів довільного стану інтерпретованої U-Y програми та запропонував метод послідовного породження шуканої множини.

Ученими інституту було розроблено ефективні алгоритми генерації інваріантів для програм. Проводились дослідження таких алгебр даних як - абсолютно вільна алгебра даних [16, 17], групи, півгрупи, абелеві группи та абелеві півгрупи, векторний простір [18], кільце многочленів [19, 20].

Методи пошуку інваріантів програм над абсолютно вільною алгеброю даних було детально вивчено та описано у працях С.Л. Кривого [16] та В.К. Сабельфельда [21]. В результаті, на базі методів потокового аналізу, побудовано повні системи перетворень для такого класу програм відносно логіко-термальної еквівалентності, введеної В.Е. Іткіним [22].

Важливим практичним та теоретичним кроком у розробці методів пошуку інваріантів є праці Халбвоша та Куазо [23], які розглядали проблему пошуку інваріантів типу лінійних рівнянь та нерівностей. Робота [24] стала розширенням алгоритму Кара [25]. Вчені застосували концепцію абстрактної інтерпретації [26] для знаходження лінійних нерівностей як інваріантів. Вони використовують методи лінійної алгебри та алгоритми лінійного програмування та запропонували алгоритми для генерування інваріантних співвідношень такого типу. Запропонований алгоритм надав можливість генерації досить багатих, хоча і неповних множин інваріантів.

Роботи [27-30] М.С. Львова стали продовженням досліджень теорії програмних інваріантів, основи якої було опубліковано в [5, 14].

В статті [27] уточнено основні задачі, які виникають в теорії інваріантів U-Y програм, описані в [14], знайдено верхню оцінку степені анулюючого полінома по кожній із змінних, що дало можливість розв'язувати задачу про співвідношення, а саме – задачу про розпізнавання факту алгебраїчної залежності набору поліномів, запропоновано алгоритм побудови «слабкого» базису ідеалу в деяких окремих ситуаціях. У [28] доведено існування нескінченних базисів множини інваріантів у випадку мови лінійних рівностей та нерівностей, а також методи побудови прикладів програм з базисами інваріантів такого роду. У статті [29] розглянуто задачу пошуку інваріантних співвідношень типу нерівностей в програмах, алгеброю даних яких є лінійно впорядковане поле. В якості моделі програми використано поняття U-Y схеми програми над пам'яттю, інтерпретованою на алгебрі даних. Продовженням досліджень з теорії інваріантів програм, алгебрами даних яких є поля, є робота [30], основними результатами якої є теореми про алгоритмічну розв'язність інваріантності даної рівності та проблеми побудови базису векторного простору всіх інваріантних рівностей, степені яких обмежено константою.

Відзначимо вклад у загальну постановку проблеми пошуку інваріантних співвідношень вчених, які працювали над дослідженням даної проблеми під керівництвом О.А. Летичевського. Зокрема, було запропоновано генерувати інваріантні співвідношення типу рівностей, нерівностей, квазірівностей та квазінерівностей з урахуванням властивостей алгебри даних, над якою працює програма. Пізніше було розроблено методи верхньої та нижньої апроксимації для генерації інваріантних співвідношень, а також відповідні алгоритми.

В [19, 31] викладено два методи побудови типів поліноміальних інваріантних рівностей у програмах, алгебра даних яких - це область цілісності (поліноміально визначені програми) або поле (раціонально

визначені програми). Один із них полягає в побудові алгебраїчних залежностей між функціями - правими частинами оператора присвоювання в тілі циклу. Другий метод - метод невизначених коефіцієнтів - будує всі інваріанти даного виду в довільній контрольній точці програми. Шаблон інваріанта задається поліноміальною формою з невизначеними коефіцієнтами. Метод заснований на властивості нетерових кілець поліномів багатьох змінних. Цю ж ідею використано в [32] для побудови поліноміальних інваріантів обмеженого степеня для програм, що визначаються поліномно. При цьому враховуються програмні умови типу $f(X) \neq 0$, дн $f(X)$ - многочлени від змінних програми. У роботі [33] запропоновано метод генерації інваріантів поліноміальних програм обмеженої степені в лінійно визначених (афінних) програмах, що містять рекурсивні виклики процедур.

Викладений в [31] загальний алгоритм обчислення поліноміальних інваріантів в довільній контрольній точці програми, засновано на ітераційному методі [14] та методі обчислення поліноміальних інваріантів обмеженої степені [30].

У [34] Deepak Kapur представив підхід до автоматичного генерування інваріантів циклічних програм, використовуючи методи елімінації кванторів. Для визначення інваріантів використовуються параметризовані формули, в яких параметри описуються генерацією обмежень на параметри, гарантуючи, що формула дійсно зберігається шляхом виконання відповідного кожному базовому циклу цикла. Підхід проілюстровано за допомогою програм, для яких твердження можна виразити у трьох пов’язаних теоріях: 1) лінійні нерівності, 2) поліноміальні рівності та 3) поліноміальні нерівності та рівності, які включають в себе 1) і 2); кожна з цих теорій використовує елімінацію кванторів.

У роботі [35] автори запропонували метод генерації інваріантів поліноміальних циклів у вигляді поліноміальних форм із застосуванням алгоритму для обчислення базисів Гребнера. Статтю [36] присвячено

алгебраїчним основам проблеми генерації поліномальних інваріантів циклів. Основним результатом дослідження є алгоритм генерації всіх поліноміальних інваріантів для циклів з так званими розв'язуваними операторами присвоювання. Зокрема, розв’язуваними є афінні оператори з позитивними реальними власними значеннями. Ті ж автори [37] запропонували спосіб генерації поліноміальних інваріантів циклів, включаючи вкладені цикли, а також враховуючи програмні умови у вигляді як поліноміальних рівностей, так і нерівностей. У роботі розглядається велика кількість прикладів та представлено таблиці для алгоритму часу залежно від технічних параметрів програми, що аналізується. У роботі [38] (2005, Kovács L. I., Jebelean T.) запропоновано алгоритм пошуку інваріантів лінійних циклів для операторів з цілими власними числами. Алгоритм виконує пошук розв’язку цієї системи, що залежить від n. Алгоритм реалізовано у програмній системі Theorema та детально проілюстровано прикладами. В [39] запропоновано новий підхід до задачі генерації поліноміальних інваріантів лінійних циклів, заснований на понятті L-інваріанта лінійного оператора. Сформульовано теорему про зв'язок L-інваріанта з мультиплікативним співвідношенням між коренями мінімального характеристичного многочлена лінійного оператора в тілі циклу, а також встановлено важливі властивості цього співвідношення у випадку, коли мінімальний характеристичний многочлен лінійного оператора непривідний над полем раціональних чисел. У [20, 40] отримано розв’язок задачі генерації поліноміальних інваріантів лінійних циклів для лінійних операторів - нетривіальних Жорданових клітинок і, в зв'язку з цим, сформульовано та вивчено поняття власного многочлена лінійного оператора. Запропоновано узагальнення теореми про зв'язок L-інваріанта з мультиплікативним співвідношенням між коренями мінімального многочлена лінійного оператора, що використовує власні многочлени лінійного оператора.

Аналіз існуючих наукових праць показує, що проблема опису інваріантних нерівностей менш вивчена. Основною складністю тут є нескінченність базису метаідеалу [41] поліноміальних нерівностей [20, 39]. Ітеративні методи вирішення задачі опису лінійних інваріантних нерівностей було запропоновано в [24, 29, 42, 43]. У роботі [24] вирішено проблему генерації найпростіших інваріантних нерівностей. У [42,43] загальні ітераційні методи використовуються для вирішення проблеми пошуку лінійних інваріантних нерівностей. У роботі [44] використано метод доведення інваріантності системи лінійних нерівностей для класу лінійних ітеративних циклів з дійсними власними числами лінійних операторів у тілі циклу. Цей метод може бути застосований до всього класу лінійних ітеративних циклів для доведення їх завершення.

Як було зазначено вище, важливість розвитку методів потокового аналізу програм полягає перш за все в їх практичній цінності. На базі алгоритмів потокового аналізу розроблено багато алгоритмів оптимізації, таких як алгоритми оптимального розподілу регістрів, редукції програм і алгоритмів з урахуванням додаткової інформації (задача змішаних обчислень [45]), алгоритми пошуку і видалення надлишкових виразів, оптимізації процедур з урахуванням входу і виходу, введення додаткових точок входу в процедури, тощо.

Після прориву в теорії верифікації, пов'язаного з появою теорії індуктивних тверджень Флойда-Хоара-Дейкстри [46] в 70-х роках, на деякий час дослідження в теорії інваріантів було частково призупинено. Але з початком інтенсивного поширення та практичного застосування засобів автоматичного доведення теорем і верифікації моделей програм замість вихідних програм, розпочався новий виток у дослідженні даної проблеми. Більшість використовуваних підходів формальних доказів властивостей програми використовують твердження, які істинні під час виконання програми в якості вхідних даних - інваріанти. Відповідно, проблема пошуку інваріантів залишається особливо актуальною і на сьогодні.

Сучасний етап досліджень з теорії програмних інваріантів почався з перевідкриття методів [47] і полягає у розширенні та удосконаленні алгоритмів генерації програмних інваріантів різних типів для різних класів об'єктних програм [33, 37, 48-51].

Розглянемо визначення поліноміального інваріанта програми [19].

Нехай A - конструктивне поле і X - множина змінних програми P. Програма P визначена поліноміально (інтерпретована над поліноміальним кільцем), якщо A - деяке поле (область цілісності), а всі її обчислюючі оператори мають вигляд

$$X := F * X,$$

де $f_i \in A(X)$, тобто многочлени з коефіцієнтами із A.

Визначення 1.1. Многочлен $g(X) \in A[X]$ називається поліноміальним інваріантом програми P, якщо

$$\{True\}\ P\ \{g(X) = 0\}.$$

Це означає, що при будь-якому початковому стані пам'яті $a =< a1, \dots, an >$, якщо програма P завершує роботу, для заключного стану пам'яті b (тобто. такого b, що $a\{P\}b$) виконується рівність $g(b) = 0$.

2.1. Алгоритми доказу та генерації поліноміальних інваріантних рівностей

У даному підрозділі розглянуто основні результати проблеми доказу та генерації поліноміальних інваріантних рівностей, представлені в [20, 39, 55].

2.1.1. L-інваріанти лінійних відображень і інваріанти лінійних циклів.

Визначення 2.1. Нехай W - n-вимірний векторний простір над полем раціональних чисел Q і $\bar{Q}$ - алгебраїчне замикання поля Q. позначимо через $X = (x_1, \dots . x_n)$ n- мірний вектор змінних. Раціональна функція $p(X) \in \bar{Q}(X)$ називається L - інваріантом лінійного оператора $A: W \to W$, якщо для будь-якого вектора $b \in W$ має місце співвідношення [20]

$$p(A \cdot b) = p(b) \qquad (2.1)$$

Приклад 2.1. (лінійний оператор з характеристичним многочленом $x^3 - 2$)

Розглянемо лінійний оператор з матрицею

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 2 & 0 & 0 \end{pmatrix},\ X = (x, y, z).$$

Раціональний вираз

$$p(x, y, z) = \frac{(\lambda_1^2 x + \lambda_1 y + z)(\lambda_3^2 x + \lambda_3 y + z)}{(\lambda_2^2 x + \lambda_2 y + z)^2} \tag{2.2}$$

де $\lambda_1 = \sqrt[3]{2},\ \lambda_2 = \sqrt[3]{2}\varepsilon,\ \lambda_3 = \sqrt[3]{2}\varepsilon^2,$ а $\varepsilon = \cos(\frac{2\pi}{3}) + i\sin(\frac{2\pi}{3})$ - первісний корінь степені 3 з 1 - L-інваріант цього оператора.

Визначення 2.2. Нехай $X = (x_1,...,x_n),\ b = (b_1,...,b_n)$ - два набори змінних. Лінійним циклом називається фрагмент імперативної програми виду

```
X := b;
While Q(X, b) do X := A*X
```

Зауваження. Оператори X:=b, X:=A*X інтерпретуються як одночасні присвоювання змінним лівих частин значень правих частин. Надалі умову Q(X, b) будемо ігнорувати, вважаючи лінійний цикл нескінченним, а його виконання недетермінованим. Т.ч. розглядаються цикли виду

```
X := b;
While True|False do X := A*X
```
(2.3)

Визначення 2.3. Нехай вектор $b^{(0)} = (b_1^{(0)},...,b_n^{(0)}) \in W$ обраний як початковий. Послідовність векторів, задана рекурентним співвідношенням $b^{(j+1)} = Ab^{(j)}$, називається орбітою лінійного оператора A з початковою точкою $\bar{b}$ і позначається через $Orbit(A,\bar{b})$.

Цикл задає орбіту лінійного оператора A в просторі W. Очевидно, орбіта A лежить в деякому одновимірному многовиді, а система інваріантів характеризує цю множину як алгебраїчну.

Визначення 2.4. Поліном $P(b,X)$ називається інваріантом циклу, якщо для будь-якого натурального j і будь-якого $b^{(0)}$ $P(b^{(0)},b^{(j)})=0$.

Теорема 2.1. Якщо $p(X)=r(X)/q(X)$ - L-інваріант лінійного оператора A, многочлен $r(X)q(b)-q(X)r(b)$ - інваріант лінійного циклу над полем $\bar{Q}$.

Такі інваріанти циклів будемо також називати *L*-інваріантами (лінійних циклів).

Приклад 2.2. (Лінійний цикл з оператором прикладу 1)

Лінійний цикл, відповідний оператору A, має вигляд

```
(x, y, z) := (a, b, c);
While True|False do (x, y, z) := (y, z, 2*x)
```

L-інваріант цього циклу визначено формулою (2.2):

$$P(x,y,z,a,b,c)=(\lambda_1^2x+\lambda_1y+z)(\lambda_3^2x+\lambda_3y+z)(\lambda_2^2a+\lambda_2b+c)^2- \\ -(\lambda_2^2x+\lambda_2y+z)^2(\lambda_1^2a+\lambda_1b+c)(\lambda_3^2a+\lambda_3b+c) \tag{2.4}$$

У [39] викладено результати, пов'язуючі L-інваріанти з власними значеннями і векторами оператора A^T. Сформулюємо основний результат цієї роботи:

Теорема 2.2 (Про мультиплікативні співвідношення). Нехай $\lambda_1,...,\lambda_m$ - власні значення лінійного оператора A і $s_1,...,s_m$ - відповідні їм власні вектори співпряженого оператора A^T. Припустимо, що існують такі цілі числа $k_1,...,k_m$, що

$$\lambda_1^{k_1}\cdot...\cdot\lambda_m^{k_m}=1\,. \tag{2.5}$$

Тоді

$$p(X)=(s_1,X)^{k_1}\cdot...\cdot(s_m,X)^{k_m} \tag{2.6}$$

- L-інваріант лінійного оператора A.

Доведення Теореми 2.2. представлено в [39].

Приклад 2.3 (продовження прикладу 2.2). Застосуємо теорему 2.2 до приклада 2.2. Обчислимо власні числа оператора *A*.

$A=\begin{pmatrix}0&1&0\\0&0&1\\2&0&0\end{pmatrix}$, $h(\lambda)=|A-\lambda E|=\begin{vmatrix}-\lambda&1&0\\0&-\lambda&1\\2&0&-\lambda\end{vmatrix}=-\lambda^3+2$. Характеристичний многочлен має вигляд $h(x)=x^3-2$. Його корені - $\lambda_1=\sqrt[3]{2},\ \lambda_2=\sqrt[3]{2}\varepsilon,\ \lambda_3=\sqrt[3]{2}\varepsilon^2$, де $\varepsilon=\cos(\frac{2\pi}{3})+i\sin(\frac{2\pi}{3})$ - первісний корінь степеня 3 з 1.

Обчислимо власні вектори s_1,s_2,s_3 матриці $A^T=\begin{pmatrix}0&0&2\\1&0&0\\0&1&0\end{pmatrix}$:

$s_1=(\lambda_1^2,\lambda_1,1),\ s_2=(\lambda_2^2,\lambda_2,1),\ s_3=(\lambda_3^2,\lambda_3,1)$.

Легко перевірити, що $\frac{\lambda_1\lambda_3}{\lambda_2^2}=1$. За теоремою 2 оператор A має L-інваріант (2.2).

Наслідок 1. Якщо мінімальний характеристичний (многочлен $h(x)$ лінійного оператора A має вільний член, рівний ±1 (тобто, $\det(A)=\pm1$), лінійний оператор A володіє L-інваріантом.

Доведення представлено в [39].

Приклад 2.4. Цикл повороту точки площини (a,b) на кут $\arctan(4/3)$.

```
(x,y) := (a,b);
While True do (x, y):= (4/5*x - 3/5*y, 3/5*x + 4/5*y)
```

Обчислимо власні значення і власні вектори оператора A:

$A=\begin{pmatrix}4/5&-3/5\\3/5&4/5\end{pmatrix}$. $h(\lambda)=|A-\lambda E|=\lambda^2-\frac{8}{5}\lambda+1$. $\lambda_1=\frac{4}{5}-i\frac{3}{5},\lambda_2=\frac{4}{5}+i\frac{3}{5}$.

$s_1=(i,1),\ s_2=(-i,1)$.

Оскільки $\lambda_1\lambda_2=1$, L-інваріант оператора A має вигляд

$p(x,y)=(ix+y)(-ix+y)=x^2+y^2$.

Інваріант циклу має вигляд $x^2+y^2-a^2-b^2$.

Приклад 2.5. Цикл обчислення послідовності Фібоначчі, починаючи з пари (a,b).

```
(x,y) := (a,b);
While True|False do (x, y):= (x + y, x)
```

Обчислимо власні значення і власні вектори оператора A:

$A=\begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}$. $h(\lambda)=|A-\lambda E|=\lambda^2-\lambda-1$. $\lambda_1=\frac{1}{2}-\frac{1}{2}\sqrt{5},\ \lambda_2=\frac{1}{2}+\frac{1}{2}\sqrt{5}$.

$s_1=(\lambda_1,1)=(\frac{1}{2}-\frac{1}{2}\sqrt{5},1),\ s_2=(\lambda_2,1)=(\frac{1}{2}+\frac{1}{2}\sqrt{5},1)$.

Оскільки $\lambda_1\lambda_2=-1$, L-інваріант оператора A має вигляд

$p(x,y)=((\lambda_1 x+y)(\lambda_2 x+y))^2=(x^2-xy-y^2)^2$.

Інваріантне співвідношення циклу має вигляд $(x^2-xy-y^2)^2=(a^2-ab-b^2)^2$.

Наслідок 2. Якщо характеристичний (мінімальний) многочлен $h(X)$ лінійного оператора A має вигляд x^m-a, лінійний оператор володіє L-інваріантами.

Доведення приведено в [39].

Такмим чином, L-інваріанти оператора A існують і обчислюються аналогічно тому, як це виконано в прикладі 3. Зокрема, L-інваріантами оператора A є раціональні вирази

$$p_i(X)=(\frac{(s_i,X)}{(s_1,X)})^{K_i},i=2,...,m,$$

де s_i- власні вектори A^*, а K_i- найменші натуральні числа такі, що iK_i кратно m: $K_i=im\ div(\gcd(i,m))$.

Теорема 2.3. Нехай $h(x)$- многочлен від змінної x з раціональними коефіцієнтами і $\Lambda=(\lambda_1,...,\lambda_m)$- всі його корені з алгебраїчного замикання $\bar{Q}$ поля Q. Розглянемо множину $G(h)=\{x_1^{k_1}\cdot...\cdot x_m^{k_m}:\lambda_1^{k_1}\cdot...\cdot\lambda_m^{k_m}=1\}$ - множина мономів з поля раціональних виразів $Q(X)$ (Можливо, з негативними

степенями), які при підстановці λ_i замість x_i набувають значення 1. Тоді $G(h)$ - мультиплікативна абелева группа зі скінченним числом утворюючих.

З теореми з випливає, що основною проблемою завдання генерації L-інваріантів є проблема пошуку алгоритму побудови множини твірних групи $G(h)$.

Приклад 2.6 (Продовження прикладу 3). Легко побачити, що для многочлена $h(x) = x^3 - 2$ мають місце наступні мультиплікативні співвідношення між його коренями:

$$\lambda_1^2 = \lambda_2\lambda_3,\ \lambda_1\lambda_2 = \lambda_3^2,\ \lambda_1\lambda_3 = \lambda_2^2,\ \lambda_2^3 = \lambda_3^3$$

Цим співвідношенням відповідають біноми

$$x_1^2 - x_2x_3,\ x_1x_2 - x_3^2,\ x_1x_3 - x_2^2,\ x_2^3 - x_3^3,$$

які утворюють базис Гребнера ідеалу $I(G_B) = I(G(h))$.

Наслідок 1. Множина всіх L-інваріантів оператора A утворює поле раціональних виразів.

Доведення є очевидним. Поле L-інваріантів оператора A, породжене елементами $G(h)$, має скінчене число твірних – элементів $M_{Gr}(h)$.

Визначення 2.5. L-інваріанти оператора A, визначені мультиплікативним співвідношенням між коренями характеристичного многочлена $\lambda_1 \cdot ... \cdot \lambda_m = \pm 1$ будемо називати цілими. L-інваріанти оператора A, визначені мультиплікативним співвідношенням $\lambda_1^{k_1} \cdot ... \cdot \lambda_m^{k_m} = 1, \sum k_i = 0$, будемо називати раціональними.

Теорема 2.4. Якщо характеристичний многочлен оператора A має вигляд $h(x^k), k > 1$, оператор A володіє раціональними L-інваріантами.

Доведення теореми наведено в [39].

Довільний невироджений лінійний оператор A у відповідному базисі може бути представлений матрицею - жордановою формою [52, 53].

$$A=\begin{bmatrix} J_1(\lambda_1) & 0 & ... & 0 \\ 0 & J_2(\lambda_2) & ... & 0 \\ . & . & ... & . \\ 0 & 0 & ... & J_m(\lambda_m) \end{bmatrix}, \tag{2.7}$$

де $J_i(\lambda_i)$ - жорданові клітинки різних розмірів. Жорданова клітинка має вигляд $J(\lambda)=\begin{bmatrix} \lambda & 1 & ... & 0 \\ 0 & \lambda & ... & 0 \\ 0 & ... & \lambda & 1 \\ 0 & ... & 0 & \lambda \end{bmatrix}$. (2.8)

Таким чином, теорема 2.2 відноситься тільки до тих рядків матриці лінійного оператора A, які відповідають власним векторам A, тобто до сукупності останніх рядків жорданових клітинок $J_i(\lambda_i)$, $i=1,...,m$. Нижче цю теорему поширено на довільні невироджені лінійні оператори.

Власні поліноми жорданових клітинок

Для аналізу лінійних циклів лінійний оператор A розглянемо як лінійне перетворення $X \leftarrow AX$ змінних $X=(x_1,...,x_n)$ лінійного простору $Q^d[|X|]$ однорідних многочленів деякої степені d. Перетворення $X \leftarrow AX$ визначає лінійне перетворення простору $Q^d[|X|]$ у себе (гомоморфізм) $T_A: Q^d[|X|] \to Q^d[|X|]$. Для $p(X) \in Q^d[|X|]$ воно задано формулою $T_A(p(X))=p(AX)$.

Визначення 2.6. Поліном $p(X) \in Q^d[|X|]$ називається власним поліномом лінійного оператора A з власним числом $\mu \in \overline{Q}$, якщо $p(X)$ - власний вектор T_A, тобто $T_A(p(X))=\mu p(X)$. Таким чином, власний поліном визначається формулою

$$p(AX)=\mu p(X). \tag{2.9}$$

Зауваження 2. Поняття власного многочлена та L-інваріанта лінійного оператора – деякі аналоги основних понять геометричної теорії інваріантів – а саме, понять відносного і абсолютного інваріанта групи G перетворень векторного простору в тому разі, коли ця група визначена як циклічна з утворюючим елементом A: $G_A=\{...,A^{-2},A^{-1},E,A,A^2,...\}$ [54].

Приклад 2.7. Нехай $\bar{s}=(s_1,..,s_n)$ - власний вектор оператора A^T та λ - його власне значення. Тоді $(\bar{s},X)=s_1x_1+...s_nx_n$ - власний поліном оператора A з власним числом λ.

Припустимо, що лінійний оператор A складається з однієї жорданової клітинки виду (2.8) розміру $k\times k$, яку позначимо через $J(k,\lambda)$. Якщо розміри клітинки і її власне значення в даному контексті не грають ролі, їх позначення ми будемо ігнорувати, позначаючи через оператор J.

Наведемо розв'язок задачі побудови всіх власних поліномів жорданової клітинки $J(k,\lambda)$. Перетворення $J:=J*X$, де $X=(x_1,...,x_k)$, в координатній формі має вигляд

$$x_1:=\lambda x_1+x_2;...;x_{k-1}:=\lambda x_{k-1}+x_k;x_k:=\lambda x_k.$$

Введемо наступні позначення: $x_{k-1}\overset{df}{=}y,\ x_k\overset{df}{=}z$. Тоді мають місце теореми:

Теорема 2.5. Не існує власних поліномів від двох змінних y,z.

Теорема 2.6. Існує система многочленів $q_j(y,z)\in U_j$, $j=1,3,...,d-1$, ступеня d і многочлен $q_{d+1}(y,z)$ ступеня $d+1$ такі, що многочлен

$$p=x_1z^d+x_2q_1(y,z)+...+x_jq_{j-1}(y,z)+...+x_dq_{d-1}(y,z)+yq_d(y,z)+q_{d+1}(y,z) \quad (2.10)$$

- власний многочлен оператора T_J.

Доказ теорем викладено у [20].

Приклад 2.8. Наведемо послідовність власних поліномів жорданової клітинки $J_5(\lambda)$ у просторі $W_5(v,w,x,y,z)$.

$\mu_1=\lambda,\ \ p_1=z$.

$\mu_2=\lambda^2,\ p_2=(x+\frac{1}{2\lambda}y)z-\frac{1}{2}y^2$.

$\mu_3=\lambda^3,\ p_3=(w-\frac{1}{3\lambda^2}y)z^2-(x)yz+\frac{1}{3}y^3$.

$\mu_4=\lambda^4,\ p_4=(v+\frac{1}{4\lambda^3}y)z^3+(-w+\frac{1}{2\lambda}x+\frac{1}{8\lambda^2}y)yz^2+(\frac{1}{2}x-\frac{1}{4\lambda}y)y^2z-\frac{1}{8}y^4$.

Теорема 2.7. Однорідний поліном степені 2 від парного числа змінних не може бути власним поліномом оператора A.

Теорема 2.8. Для жорданової клітинки $J(k,\lambda)$ існує набір B_n, що складається з $k-1$ власного полінома вигляду (2.10): $B_k =< p_1, p_2,..., p_{k-1} >$,

$p_1 = z,\ \ p_2 \in Q(\lambda)[z, y, x_{k-2}],\ \ \ldots,\ \ p_{n-1} \in Q(\lambda)[z, y, x_1,..., x_{k-2}]$.

Будь-який власний поліном $q(X)$ жорданової клітинки $J_n(\lambda)$ можна представити у вигляді

$$q(X) = \frac{1}{z^k} F(p_1, p_2,...p_{k-1}). \qquad (2.11)$$

де F - многочлен від $k-1$ змінних з коефіцієнтами з $Q(\lambda)$.

Доведення теореми наведено в [20].

Матричне формулювання задачі та алгоритм обчислення власних поліномів описано в [20].

2.1.2. Власні поліноми та L-інваріанти лінійних операторів

Нехай A - довільний невироджений лінійний оператор, який діє на векторному просторі $Q^{(d)}[X]$ однорідних многочленів як лінійне перетворення змінних $f(X) \to f(AX)$, d - натуральне число і $X' \subseteq X$. Множина власних поліномів степеня d від змінних з X' утворює скінченовимірний векторний підпростір, який ми позначимо через $W(A,d,X')$. Базис цього підпростору складається зі скінченого числа поліномів $(q_1,..,q_M)$.

Для операторів – жорданових клітинок теорема 2.5 дає опис цього базису через поліноми набору $(p_1,..., p_{k-1})$. Саме,

$q_j(X) = \frac{1}{p_1^k} F_j(p_1, p_2,...p_{k-1})$, $F_j(u_1,...,u_{n-1}) = a_{1j}M_{1j},+...+a_{K_j j}M_{K_j j}$, $a_{ij} \in Q(\lambda)$, M_{ij} - мономи від змінних $(u_1,...,u_{n-1})$.

Для кожної жорданової клітинки $J_k(\lambda_k)$ жорданової форми оператора A визначена своя послідовність підпросторів власних поліномів. Нехай J - одна така клітинка. Тоді, у визначеннях, використовуваних вище при розгляді операторів – жорданових клітинок, ця послідовність має вигляд $W(J,1,(z)) \subset W(J,3,(x_{k-1}, y, z)) \subset ... \subset W(J,k,(x_1,..., x_{k-1}, y, z))$.

Власним числом підпростору $W(J(\lambda),d,X^{'})$ назвемо число λ^d. Послідовність власних чисел розглянутих підпросторів має вигляд $(\lambda,\lambda^3,...,\lambda^k)$ Якщо жорданова форма оператора A складається з декількох клітинок, у визначенні підпросторів власних поліномів A має сенс розглядати тільки підмножини змінних виду $X^{'}=\bigcup_J X_i^{(J)}$, где $X_i^{(J)}\overset{df}{=}\{x_i^{(J)},...,x_{n_j-1}^{(J)},y^{(J)},z^{(J)}\},i\geq 1$ - підмножини змінних, відповідних даній жордановій клітинці оператора A, а об'єднання береться по всім жордановым клітинкам оператора A.

Розглянемо лінійний оператор, утворений двома жордановими клітинками $J_{n_1}(\lambda_1),J_{n_2}(\lambda_2)$. Будь-який підпростір власних поліномів оператора A в цьому випадку є прямим добутком підпросторів жорданових клітинок:

$$W(J_{n_1},d_1,X_1)\times W(J_{n_2},d_2,X_2)=W(A,d_1+d_2,X_1\cup X_2).$$

Через $Base(W)$ позначимо базис підпростору W. Тоді

$$Base(W(J_{n_1},d_1,X_1)\times W(J_{n_2},d_2,X_2))=Base(W(J_{n_1},d_1,X_1))\times Base(W(J_{n_2},d_2,X_2)).$$

Якщо

$$Base(W(J_{n_1},d_1,X_1))=(q_{11}(X_1),...,q_{1k_1}(X_1)),$$

$$Base(W(J_{n_2},d_2,X_2))=(q_{21}(X_2),...,q_{2k_2}(X_2)),$$

то

$$Base(W(J_{n_1},d_1,X_1)\times W(J_{n_2},d_2,X_2))=\{q_{1i}(X_1)q_{2j}(X_2),\ i=1..k_1,\ j=1..k_2\},$$

а власне число цього підпростору дорівнює $\lambda_1^{d_1}\lambda_2^{d_2}$. Ця конструкція безпосередньо поширюється на оператори, що містять довільну кількість жорданових клітинок довільних розмірів.

Якщо жорданова форма лінійного оператора A складається з жорданових клітинок $J(\lambda_1,n_1,X_1),...,J(\lambda_m,n_m,X_m)$, тобто $A=J(\lambda_1,n_1,X_1)\times...\times J(\lambda_m,n_m,X_m)$, для оператора A можна виділити систему підпросторів власних поліномів, кожен з яких характеризується власним числом $\lambda_1^{d_1}\lambda_2^{d_2}...\lambda_m^{d_m}$, $d_j\leq n_j, j=1,...,m$.

Теорема 2.9. Якщо власні числа двох підпросторів власних поліномів лінійного оператора A рівні, їх сума також утворює підпростір власних поліномів: $\lambda_1^{k_1}\lambda_2^{k_2}...\lambda_m^{k_m} = \lambda_1^{l_1}\lambda_2^{l_2}...\lambda_m^{l_m} = \mu \Rightarrow W(\mu, X_1) + W(\mu, X_2) = W(\mu, X_1 \cup X_2)$.

Доведення є очевидним.

Нижче ми покажемо, що визначення підпросторів власних поліномів – одна з найбільш важливих задач теорії програмних інваріантів лінійних операторів.

Розглянемо тепер задачу побудови L - інваріантів лінійних операторів (див. Визначення 2.1). Нехай оператор A складається з однієї жорданової клітинки і $q(X)$ - власний поліном A з власним числом λ^d. Тоді, очевидно, раціональний вираз $r(X) = q(X)/z^d$ - L - инваріант A. Таким чином, для жорданової клітинки розміру d існує принаймні $d-2$ L - інваріанта

$$r_3(X) = \frac{p_3(X)}{z^3}, ..., r_n(X) = \frac{p_d(X)}{z^d}. \qquad (2.12).$$

Ці інваріанти ми будемо називати внутрішньоклітинковими.

Зауваження. Система L-інваріантів (2.11), що визначається через власні многочлени, задає т. н. орбіту лінійного оператора A у просторі W. Дійсно, якщо вектор $b^{(0)} = (b_1^{(0)}, ..., b_n^{(0)})$ обраний в якості початкового, послідовність векторів, задана рекурентним співвідношенням $b^{(j+1)} = Ab^{(j)}$, лежить у двовимірному алгебраїчному многовиді, заданому системою рівнянь

$$[r_3(X) = r_3(b^{(0)})] \& ... \& [r_n(X) = r_n(b^{(0)})] \qquad (2.13)$$

Можна очікувати, що орбіта A, як нескінченна послідовність, лежить в одновимірному різноманітті, причому відсутнім членом системи (2.13) має бути рівняння, яке задається L- інваріантом, залежним від y, z.

Теорема 2.5 стверджує, що таких інваріантів не існує. Однак, легко перевірити, що неалгебраїчна функція $p_{yz} = y - \frac{1}{\lambda \ln(\lambda)} z \ln(z)$ задовольняє співвідношенню (2.9): $p_{yz}(AX) = \lambda P_{yz}(X)$. Тому до алгебраїчної системи (2.13)

можна додати неалгебраїчне рівняння, отримавши опис одновимірної множини, що містить орбіту A:

$$[b_n^{(0)} p_{yz}(y,z) = zp_{yz}(b_{n-1}^{(0)}, b_n^{(0)})] \& [r_3(X) = r_3(b^{(0)})] \& \ldots \& [r_n(X) = r_n(b^{(0)})].$$

Теорема 2.9 визначає так звані міжклітинкові інваріанти. Дійсно, нехай простір власних многочленів довільного лінійного оператора містить два різних підпростори W_1, W_2 з рівними власними значеннями (тобто задовольняють теоремі 2.7), $q_1(X_1) \in W_1$, $q_2(X_2) \in W_2$. Тоді $r(X_1 \cup X_2) = q_1(X)/q_2(X)$ - L - інваріант оператора A.

Теорема 2.10. Нехай $r(X) = q(X)/s(X)$ - L- інваріант лінійного оператора A. Тоді $q(X), s(X)$ - власні поліноми оператора A з рівними власними числами.

Доведення теореми наведено в [20].

Теорема 2.11. Нехай $q_1,...,q_m$ - множина всіх базисних поліномів усіх жорданових клітинок лінійного оператора A та $\mu_1,...,\mu_m$ - сукупність їх власних чисел. Припустимо, що існують такі цілі числа $k_1,...,k_m$, що

$$\mu_1^{k_1} \cdot \ldots \cdot \mu_m^{k_m} = 1 \ . \qquad (2.14)$$

Тоді, $r(X) = q_1^{k_1} \cdot \ldots \cdot q_m^{k_m}$ - L- інваріант лінійного оператора A.

Доказ є очевидним.

Таким чином, проблема опису L - інваріантів лінійного оператора A зводиться до проблеми опису всіх співвідношень (2.14). У [39] доведено, що ця множина має скінчений базис.

Якщо всі власні числа лінійного оператора A - раціональні числа, проблема побудови цього базису алгоритмічно розв'язна з допомогою теоретико-числового алгоритму.

У разі, коли λ_j - алгебраїчні числа, проблема побудови базису множини співвідношень (2.5) залишається відкритою.

У статті [55] описано прямий метод знаходження інваріантів жорданових клітинок без спирання на власні многочлени. Основні результати цієї роботи розглянуто нижче.

Теорема 2.12 (про структуру ідеалу інваріантів). Нехай A – довільний невироджений лінійний оператор, представлений у відповідному базисі матрицею (2.7), $I_{J_1}(A),...,I_{J_k}(A)$ – ідеали інваріанти його жорданових клітинок, представлені в однорідних координатах

$u_{ij} = x_{ij}/z_i,\ u_i = y_i/z_i \qquad e_{ij} = a_{ij}/c_i, e_i = b_i/c_i$

базисами виду

$u_j - q_j(\lambda, u, 1)$, $j = 1,...,n-2$

та $I_\Lambda(A)$ - ідеал інваріанту оператора A_{red}, і $I(A)$ - ідеал інваріанти оператора A (цикла (2.3)). Тоді

$$GBase(I_\Lambda(A)) = GBase(I_{J_1}(A)) \cup ... \cup GBase(I_{J_k}(A)) \cup GBase(I_\Lambda(A)).$$

Теорема 2.13. Якщо група мультиплікативних співвідношень коренів непривідного многочлена $f(x)$ нетривіальна ($MR(f) \neq (e)$), можуть мати місце дві ситуації:

1. Множину коренів $\Lambda = (\lambda_1,...,\lambda_n)$ розбито на деяке число l рівних класів $\Lambda_1,...,\Lambda_l$; $\Lambda_j = \{\lambda_{(j-1)d+1},...\lambda_{jd}\}$; $j = 1,...,l.$ При цьому $d = len(\Lambda_j), n = ld$. Мультиплікативні співвідношення з $MR(f)$ у цій ситуації мають вигляд $\Lambda_j = \varepsilon_j \Lambda_j$, $j = 1,...,l$, де ε_j - корені з 1.

2. Множина коренів $\Lambda = (\lambda_1,...,\lambda_n)$ розбивається на деяку кількість k рівнопотужних класів $\Lambda_1,...,\Lambda_l, \Lambda_i = \{\lambda_{(i-1)d+1},...\lambda_{id}\}; i = 1,...,k.$ При цьому $d = len(\Lambda_j), n = kd$. Мультиплікативні співвідношення з $MR(f)$ у цій ситуації мають вигляд $\Lambda_i = \varepsilon_{ij}\Lambda_j$, $i = 1,...,l$, де ε_j – корені з 1.

Обидві ситуації можуть мати місце одночасно.

Доказ теореми 2.13 див. у [55].

Ця теорема грає ключову роль для алгоритму обчислення системи утворючих групи $MR(f)$.

Теорема 2.14. Нехай $f(x) \in Q[x]$ - непривідний многочлен, $\lambda_1,...,\lambda_m$ - його корені. Проблема побудови базису множини утворюють групи $G_U(h) = \{x_1^{k_1}...x_m^{k_m} : \lambda_1^{k_1}...\lambda_m^{k_m} \in U\}$, де U - група всіх коренів з одиниці, алгоритмічно розв'язна.

Доказ теореми 2.14 див. у [55].

2.2. Алгоритми доказу інваріантних нерівностей

Нехай $W = K^n$ - n-вимірний векторний простір над лінійно-впорядкованим конструктивним полем K і $\bar{K}$ - алгебраїчне замикання K.

Визначення 2.7. Лінійною напівалгебраїчною множиною $M(x_1,...,x_n)$ називається область W, визначена бесквантерною формулою в сигнатурі логічних зв'язок $<\vee,\&,\neg>$ з лінійними нерівностями від змінних $x_1,...,x_n$ в якості атомів. Якщо область задається формулою $F(X)$, тобто $M = \{X : F(X)\}$, позначатимемо її через $M(F(X))$.

Визначення 2.8. Нехай $X = (x_1,...,x_n)$, $\bar{b} = (b_1,...,b_n)$ - два вектора змінних. Лінійно визначеним циклом з передумовою називається фрагмент імперативної програми виду

```
X := b;   // S(b̄) - передумова
While U(X, b) do X := A*X                    (2.15)
```

де $S(\bar{b})$, $U(X,\bar{b})$ – безквантерні формули прикладної логіки лінійних напівалгебраїчних многовидів, A - матриця лінійного оператора $W \to W$.

Недетермійованим циклом, асоційованим з циклом (2.15), називається цикл вигляду

```
X := b;    // S(b̄) - передумова
While True|False do X := A*X                 (2.16)
```

кількість повторень якого недетермінована.

Зауваження. Визначення 2.8 циклів відрізняється від визначення 2.2 наявністю передумови $S(\bar{b})$, обмежуючою початкові значення змінних циклу

лінійною півалгебраїчною множиною та введенням у розгляд умови циклу $U(X,\bar{b})$.

Визначення 2.9. Лінійна нерівність $P(X,b) \in K^1[X,b]$ називається інваріантною для циклу (2.15) з передумовою , якщо вона виконується кожного разу в результаті виконання тіла циклу. $P(X,b) \overset{df}{=} a_1x_1 + ... + a_nx_n < a_1'b_1 + ... + a_n'b_n$

Таким чином, інваріантність означає виконання послідовності формул

$S(b) \to P(b,b)$, // інваріант виконується при вході в цикл

$U(b,b) \to P(Ab,b)$, // інваріант виконується після 1-ої ітерації

$U(Ab,b) \to P(A^2b,b)$, // інваріант виконується після 2-ої ітерації

…

$U(A^kb,b) \to P(A^{k+1}b,b)$, // інваріант виконується після до-ої ітерації

$\neg U(A^kb,b) \to P(A^kb,b)$ // інваріант виконується при завершенні циклу

Теорема 2.15. Якщо всі власні значення $\Lambda = (\lambda_1,...,\lambda_n), \lambda_i \in \overline{K}$ оператора A дійсні, проблема доказу інваріантності $P(X,b)$ для циклу (2.15) є алгоритмічно розв'язуваною.

Доведення. Доказ інваріантності системи лінійних нерівностей очевидним чином зводиться до доказу інваріантності кожної із нерівностей системи. Основний зміст доведення теореми сформульовано в лемах 2.3-2.7.

Розглянемо лінійну нерівність з раціональними коефіцієнтами

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n \leq c, c = c(b). \qquad (2.17)$$

Задача полягає в тому, щоб довести або спростувати інваріантність цієї нерівнсті для циклу (2.15). Припустимо, що лінійний оператор в тілі циклу діагоналізований. У базисі власних векторів його матриця та її $m-$ а степінь матимуть вигляд:

$$A = \begin{bmatrix} \lambda_1 & 0 & ... & 0 \\ 0 & \lambda_2 & ... & 0 \\ ... & ... & ... & ... \\ 0 & 0 & ... & \lambda_n \end{bmatrix}, A^m = \begin{bmatrix} \lambda_1^m & 0 & ... & 0 \\ 0 & \lambda_2^m & ... & 0 \\ ... & ... & ... & ... \\ 0 & 0 & ... & \lambda_3^m \end{bmatrix}.$$

Зауваження. Після переходу до базису із власних векторів коефіцієнти нерівності зміняться. Якщо $S(\Lambda)$ – матриця переходу, то нові значення векторів a, b розраховіюються за формулами $a^{(S)} = SaS^{-1}, b^{(S)} = SbS^{-1}$.

Лема 2.1. Твердження теореми справедливо для циклу (2.16) при $S(b) = \{b\}$.

Доведення. Нехай b – вектор із Q^n. Припустимо, що $S(b) = \{b\}$. Перед входом в цикл інваріантність (2.17) означає, що

$$a_1 b_1 + a_2 b_2 + \cdots + a_n b_n \leq c. \qquad (2.18)$$

Після m ітерацій циклу інваріантність (18) означає, що

$$\lambda_1^m a_1 b_1 + \lambda_2^m a_2 b_2 + \cdots + \lambda_n^m a_n b_n \leq c. \qquad (2.19)$$

Без обмеження загальності можна вважати, що $|\lambda_1| > \max(|\lambda_2|, \ldots, |\lambda_m|)$, а також $a_1 \neq 0, b_1 \neq 0$, оскільки в іншому випадку змінна x_1 не приймає участь в розрахунках. Розглянемо різні випадки:

1. Нехай $|\lambda_1| \leq 1$. Відображення $A: Q^n \to Q^n$ зтискаюче.

1.1. Якщо $\lambda_1 > 0$, нерівність (2.18) перепишемо у вигляді

$$a_1 b_1 + (\lambda_2^m / \lambda_1^m) a_2 b_2 + \cdots + (\lambda_n^m / \lambda_1^m) a_n b_n \leq c / \lambda_1^m . \qquad (2.20)$$

Оскільки $\lambda_j^m / \lambda_1^m \to 0$, $c / \lambda_1^m \to \infty$, при $c > 0$ існує таке натуральне M_0, що при $m > M_0$ (20) нерівність виконується. При $c < 0$ існує таке натуральне M_0, що при $m > M_0$ нерівність (2.20) хибна. Відповідно в обох випадках інваріантність (2.17) можна встановити скінченою кількістю $m \leq M_0$ перевірок.

1.2. Якщо $\lambda_1 < 0$, нерівність (2.20) перепишемо у вигляді системи нерівностей – парній та непарній кількості повторень циклу

$$a_1 b_1 + (\lambda_2^{2m} / \lambda_1^{2m}) a_2 b_2 + \cdots + (\lambda_n^{2m} / \lambda_1^{2m}) a_n b_n \leq c / \lambda_1^{2m},$$

$$a_1 b_1 + (\lambda_2^{2m+1} / \lambda_1^{2m+1}) a_2 b_2 + \cdots + \left(\frac{\lambda_n^{2m+1}}{\lambda_1^{2m+1}}\right) a_n b_n \geq \frac{c}{\lambda_1^{2m+1}}. \qquad (2.21)$$

При $c > 0$ існує таке натуральне M_0, що при $m > M_0$ система (2.21) виконується. Відповідно, в цьому випадку інваріантність (2.17) можна встановити скінченою кількістю $m \leq M_0$ перевірок. При $c < 0$ система (2.21) хибна.

2. Будемо вважати, що $|\lambda_1| > 1$. Як і в п.1, розглянемо декілька випадків.

2.1. Нехай $\lambda_1 > 0$. Приведемо нерівність (20) до вигляду

$$a_1 b_1 + (\lambda_2^m/\lambda_1^m)a_2 b_2 + \cdots + (\lambda_n^m/\lambda_1^m)a_n b_n \leq c/\lambda_1^m \quad (2.22)$$

Далі,

$$a_1 b_1 \leq (c/\lambda_1^m) - (\lambda_2^m/\lambda_1^m)a_2 b_2 - \cdots - (\lambda_n^m/\lambda_1^m)a_n b_n \quad (2.23)$$

Оскільки $|\lambda_j/\lambda_i| < 1$ і $|\lambda_1| > 1$, $\lambda_j^m/\lambda_1^m \xrightarrow[m\to\infty]{} 0$, $c/\lambda_1^m \xrightarrow[m\to\infty]{} 0$, при достатньо великих значеннях m і кожен доданок правої частини, і модуль правої частини нерівності можуть бути як завгодно малими:

$$|(c/\lambda_1^m) - (\lambda_2^m/\lambda_1^m)a_2 b_2 - \cdots - (\lambda_n^m/\lambda_1^m)a_n b_n| < \varepsilon.$$

Таким чином, при $m > M_0(\varepsilon)$ маємо

$$|a_1 b_1| \leq \varepsilon. \quad (2.24)$$

2.1.а. Відповідно, якщо $a_1 b_1 > 0$, ε можна вибрати настільки малим, що $a_1 b_1 > \varepsilon > 0$. Це суперечить (2.20). Значить, нерівність (2.17) не є інваріантною, оскільки не виконується при $m > M_0(\varepsilon)$.

2.1.б. Якщо ж $a_1 b_1 < 0$, нерівність (2.17) виконується при всіх $m > M_0(\varepsilon)$, і, відповідно, може бути встановлена перевірками скінченного числа M співвідношень, що визначають інваріантність нерівності з Визначення 2.9.

3. Розглянемо випадок $\lambda_1 < 0$. Аналогічно п. 1.2., при парних m має місце нерівність (21), при непарних m – нерівність

$$a_1 b_1 \geq (c/\lambda_1^m) - (\lambda_2^m/\lambda_1^m)a_2 b_2 - \cdots - (\lambda_n^m/\lambda_1^m)a_n b_n,$$

тобто $a_1 b_1 \geq \varepsilon$. Оскільки в циклі парні та непарні значення m чергуються, (2.17) не є інваріантом.

Лема 2.2. Твердження теореми справедливо для циклу (2.16) з обмеженою опуклою множиною в якості передумови: $S(b) = BoundPolygon(b^{(1)},...,b^{(k)})$.

Доведення. Розглянемо випадок, коли $S(b)$ – обмежена багатогранна область Q^n, натягнута на вектори $b^{(1)}, \ldots, b^{(k)}$. У цьому випадку будь-який вектор із $S(b)$ можна представити у вигляді

$$b = a_1 b^{(1)} + \cdots + a_k b^{(k)}, a_j \geq 0, a_1 + \cdots + a_k = 1.$$

Тоді випадок 2.1.б повинен мати місце для кожного $b(j), j = 1, \dots, k$: при $m_j > M_{0j}(\varepsilon)$ виконується $a_1 b_1^{(j)} < 0$. При цьому із $b = a_1 b^{(1)} + \cdots + a_k b^{(k)}$ слідує, що $b_1 = a_1 b_1^{(1)} + \cdots + a_k b_k^{(k)}$ і при $m > \max[M_{01}(\varepsilon), \dots, M_{0k}(\varepsilon)]$ має місце $a_1 b_1 < 0$.

Лема 2.3. Твердження теореми справедливо для циклу (2.16) з необмеженою випуклою множиною в якості передумови: $S(b) = UnboundPolygon(b^{(1)},\dots,b^{(k)})$.

Доведення. Уявімо, що область $S(b)$ не обмежена. Обчислимо $Max = \max(P(x, b), X \in S(b))$. Цей максимум або є скінченним і досягається в одній з вершин $S(b)$, або нескінченний. Обчислення Max здійснюється методами лінійного програмування. Для інваріантності (2.18) необхідно щоб $Max <$ c при розрахунках, наведених в Лемі 2.2.

Для повноти доказу у випадку діагоналізуємої матриці оператора A залишилося розглянути випадок, коли декілька модулів власних значень приймають максимальні значення:

$|\lambda_1| = \cdots = |\lambda_k| > \max(|\lambda_{k+1}|, \dots, |\lambda_n|)$.

Нерівність (2.19) має вигляд $a_1 b_1 + \cdots + a_k b_k \leq \varepsilon$ і всі міркування залишаються такими ж.

Лема 2.4. Твердження теореми циклу (2.16) справедливо у випадку, коли оператор в жордановій формі має нетривіальні жорданові клітинки.

Доведення. Нехай матриця оператора A в жордановій формі має нетривіальні жорданові клітинки, тобто має вигляд $A = J_1(\lambda_1) \times \dots \times J_l(\lambda_l)$. Жорданові клітинки – це матриця $n \times n$:

$$J(\lambda) = \begin{bmatrix} \lambda & 1 & \dots & 0 \\ 0 & \lambda & \dots & 0 \\ 0 & \dots & \lambda & 1 \\ 0 & \dots & 0 & \lambda \end{bmatrix} \tag{2.25}$$

Усі жорданові клітинки розміщено по головній діагоналі, і кожна з них визначається своїм власним значенням оператора A.

Кожній клітинці $J_i(\lambda_i)$ відповідає власна група змінних. Таким чином, доведення достатньо провести лише для однієї клітинки розміру k.

Нехай $X = (x_1, \dots, x_k)$, m – натуральне число і $b = (b_1, \dots, b_k)$ – вектор змінних (початкових значень). Обчисливши m-ту операцію $X^{(0)} = b$; $X^{(m)} = AX^{(m)}$ у явному вигляді $X^{(m)} = A^{(m)}X$: $X^{(m)} = J(\lambda^{(m)}b)$, отримаємо

$$X^{(m)} = \begin{bmatrix} \lambda^m & C_1(m) & \dots & C_{k-1}(m) \\ 0 & \lambda^m & \dots & C_{k-2}(m) \\ 0 & \dots & \lambda^m & C_1(m) \\ 0 & \dots & 0 & \lambda^m \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_k \end{bmatrix}, \qquad (2.26)$$

де $C_j(m) = C_m^j \lambda^{m-j} = \frac{m(m-1)\dots(m-j+1)}{j!}\lambda^{m-j}, j \in 1 \dots k-1.$

Розглянемо рівність, що відповідає j-тому рядку матричної рівності

$$x_j^{(m)} = \lambda^m \left(b_j + \frac{C_1(m)}{\lambda} b_{j+1} + \dots + \frac{C_{k-j}(m)}{\lambda^{k-j}} b_k \right) \stackrel{\text{def}}{=} \lambda^m g_i(m, \lambda, b). \quad (2.27)$$

Ліва частина нерівності (2.19) являє собою суму, кожен доданок якої визначено групою змінних клітинки $J_i(\lambda_i)$:

$$\lambda_j^m S_j^{(m)} = \lambda_j^m \left(g_1(m, \lambda_j, b_{j1}) a_{j1} + g_2(m, \lambda_j, b_{j2}) a_{j2} + \dots + g_k(m, \lambda_j, b_{jk}) a_{jk} \right). \quad (2.26)$$

Аналог нерівності (2.19) має вигляд

$$\lambda_1^m S_1^{(m)} + \lambda_2^m S_2^{(m)} + \dots + \lambda_l^m S_l^{(m)} \le c. \qquad (2.28)$$

Нехай λ_1 – максимальне значення: $|\lambda_1| = \max(|\lambda_1|, \dots, |\lambda_l|)$. Тоді, оскільки $S_j^{(m)}$ - многочлен від m - а $(\lambda_j/\lambda_1)^m$ показова функція від m і $\lambda_j/\lambda_1 > 1$, маємо $\frac{\lambda_j^m S_j^{(m)}}{\lambda_1^m} \xrightarrow[m\to\infty]{} 0$ при $\lambda_j \ne \lambda_1$.

Таким чином, метод, описаний в Лемах 2.1-2.3, можна використовувати і для (24), тобто в загальному випадку. Не повторюючи розгляд усіх випадків леми, обмежимося випадком $\lambda_1 > 1$. Як і в лемі 2.1, необхідною умовою існування інваріанта $S_1^{(m)} < 0$ запишемо

$$S_1^{(m)} = \left(g_1(m, \lambda_1, b_{11}) a_{11} + g_2(m, \lambda_1, b_{21}) a_{12} + \dots + g_k(m, \lambda_1, b_{1k_1}) a_{1k_1} \right),$$

$S_1^{(m)}$ - многочлен від змінної m.

Для того, щоб забезпечити нерівність $S_1^{(m)} < 0$ при всіх $m > M_0$, необхідно, щоб коефіцієнт $Lc\left(S_1^{(m)}\right)$ при старшій степені $S_1^{(m)}$ був меньше нуля. З (2.26), (2.27) слідує, що $Lc\left(S_1^{(m)}\right) = Lc\left(C_{k_1}(m)a_{1k_1}b_{1k_1}\right)$, але $Lc\left(C_{k_1}(m)\right) = m^{k1}/\lambda_1^{k1} > 0$. Тому $a_{1k_1}b_{1k_1} < 0$ – необхідна умова інваріантності нерівності. Для циклу (2.16) теорему доведено.

Лема 2.5. Твердження теореми справедливо для циклу (2.15) (Рис.2.1).

Аналіз циклу (2.15), ілюстрований блок-схемою рис. 2.1, показує, що образ множини $S(b)$ після m кратного виконання циклу в контрольній точці С1 описується системою нерівностей $s(A^m b) \& U(A^m b, b)$. Отже, метод лем 2.1-2.4, застосований до передумови $s(b) \& U(b,b)$ вирішує завдання доказу інваріантності нерівності в контрольній точці С1 .

Доказ.

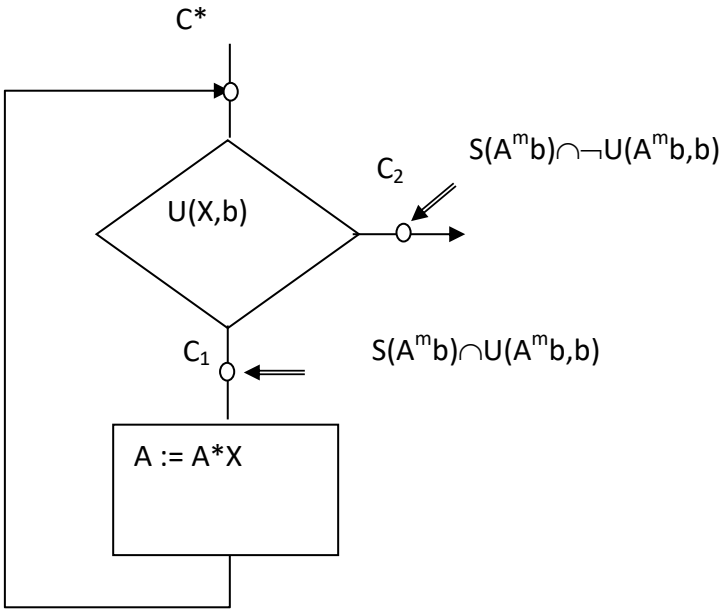


Рис.2.1. *Цикл (2.15) та інваріанти його контрольних точок.*

Таким чином, існує таке натуральне число M_0, що якщо кількість m повторень циклу перевершує M_0 і $U(X,b)$ виконується для $k = 0,1,...,M_0$, то $U(X,b)$ виконується при будь-якому $m > M_0$. Теорему доведено.

Визначення 2.10. Послідовність $\{\bar{b}^{(m)}\}$ визначена рекурсивними співвідношеннями

$$\bar{b}^{(0)} = \bar{b}, \bar{b}^{(m+1)} = A\bar{b}^{(m)}, m = 0,1,... \qquad (2.29)$$

Називається орбітою лінійного оператора A з початковою точкою $\bar{b}$, і позначається $Orbit(A, \bar{b})$.

Визначення 2.11. Лінійно визначений цикл (2.15) називається завершуваним, якщо для будь-якого $\bar{b} \in M(S(X))$ послідовність (2.29) при деякому натуральному $m^* = m^*(\bar{b})$ задовольняє співвідношенню $\neg U(\bar{b}^{(m^*)}, \bar{b})$.

Таким чином, якщо цикл є завершуваним, для кожного $\bar{b} \in M(S(X))$ існує найменше натуральне число $m^*(\bar{b})$, на якому цикл (16) завершується.

Визначення 2.12. Нехай $\bar{a}, \bar{c} \in K^n$. Лінійна нерівність

$$L(\bar{a}, \bar{c}, X, \bar{b}) \overset{df}{=} (\bar{a}, X) \le (\bar{c}, \bar{b}) \qquad (2.30)$$

називається умовним інваріантом лінійно визначеного циклу (2.15) (з передумовою $S(\bar{b})$), якщо для будь-якого $\bar{b} \in M(S(X))$ $Orbit(A,\bar{b})$ (див. визн. 2.10) задовольняє співвідношенням

$$S(\bar{b}) \to L(\bar{a}, \bar{c}, \bar{b}, \bar{b}),$$

$$U(\bar{b}^{(m-1)}, \bar{b}) \to L(\bar{a}, \bar{c}, \bar{b}^{(m)}, \bar{b}),\ m = 1,2,...,m^*(\bar{b}).$$

Зауваження. Якщо цикл (2.15) не завершується (розбігається) в деякій точці $\bar{b}$, $m^*(\bar{b})$ слід вважати рівним нескінченності: $m^*(\bar{b}) = +\infty$.

Метод доказу інваріантності лінійних нерівностей для лінійних операторів з комплексними власними числами.

Приклад 2.9.

Нехай

$$S(x, y) = (0 \le x \le 1) \,\&\, (0 \le y \le 1),$$

$$U(x, y, b_1, b_2) = \neg(|x + b_1| \le \varepsilon) \,\&\, (|y + b_2| \le \varepsilon),$$

$$A = \begin{bmatrix} 3/5 & 4/5 \\ -4/5 & 3/5 \end{bmatrix}.$$

$$L = x + y \le 2b_1 + 2b_2 \,//\, \bar{a} = (1,1),\ \bar{c} = (2,2).$$

У цьому прикладі лінійний оператор A - оператор повороту на кут $\alpha = arctg(4/3)$. Початкова точка $\bar{b}$ належить одиничному квадрату. Орбіта

лінійного оператора A - послідовність, кожна точка якої лежить на окружності $x^2 + y^2 = b_1^2 + b_2^2$.

Умова повторення циклу – «точка (x, y) лежить поза квадратом зі стороною 2ε і центром в точці $(-b_1, -b_2)$». Тому цикл завершиться, коли точка потрапить всередину цього квадрата, тобто точка зробить поворот на кут $\pi + 2k\pi$ з точністю ε. Оскільки кут α неспівмірний з π, орбіта оператора A - усюди щільна множина на окружності $x^2 + y^2 = b_1^2 + b_2^2$, - цикл завершиться.

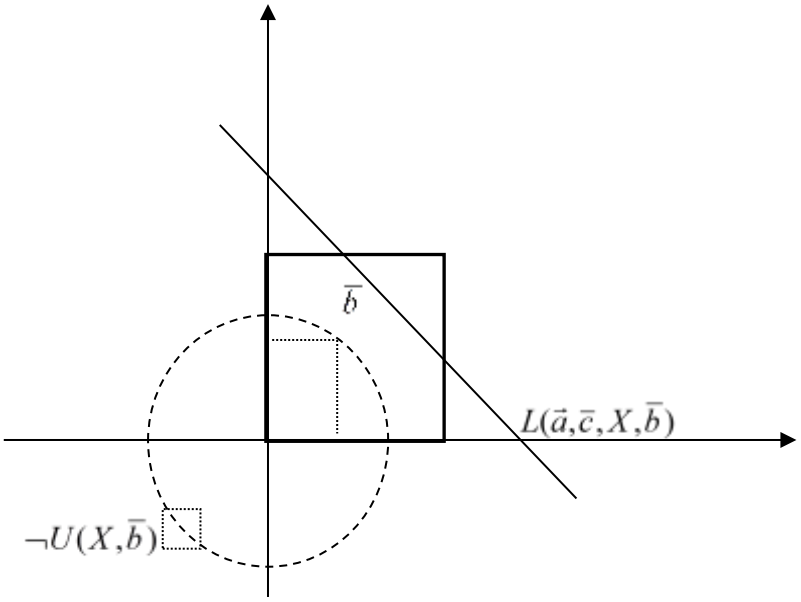


Рис.2.2. *Геометрична ілюстрація лінійно-визначеного циклу.*

Нехай $f(x)$ - мінімальний характеристичний многочлен оператора A, $\Lambda = \{\lambda_1, ..., \lambda_n\}$ - множина його коренів (спектр A) та $\bar{e}_1, ..., \bar{e}_n$ - множина власних векторів A. Нехай, далі, $\lambda_1, ..., \lambda_{2k}$ - множина комплексних власних чисел, а $\lambda_{2k+1}, ..., \lambda_n$ - множина дійсних власних чисел, причому $\lambda_1 = \overline{\lambda}_2, ..., \lambda_{2k-1} = \overline{\lambda}_{2k}$. Визначимо вектори

$$\bar{l}_1 = \frac{1}{2}(\bar{e}_1 + \bar{e}_2),\ \bar{l}_2 = \frac{1}{2i}(\bar{e}_1 + \bar{e}_2).$$

В базисі $(\bar{l}_1, \bar{l}_2)$ матриця $A_1 = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \overline{\lambda}_1 \end{bmatrix}$ має вигляд $B_1 = \begin{bmatrix} \alpha_1 & \beta_1 \\ -\beta_1 & \alpha_1 \end{bmatrix}$, де $\lambda_1 = \alpha_1 + i\beta_1,\ \lambda_2 = \alpha_1 - i\beta_1$. Застосувавши таке перетворення до всіх пар власних

векторів з попарно співпряженими комплексними власними числами, отримаємо представлення лінійного оператора в так званій дійсній жордановій формі:

$$A' = \begin{bmatrix} B_1 & 0 & . & 0 & . & . & 0 \\ 0 & B_2 & . & 0 & . & . & 0 \\ . & . & . & . & . & . & . \\ 0 & ... & 0 & B_k & . & ... & 0 \\ 0 & . & . & 0 & \lambda_{2k+1} & ... & 0 \\ . & . & . & . & . & . & . \\ 0 & . & . & . & 0 & ... & \lambda_n \end{bmatrix}$$

Зауваження. Після переходу до базису з власних векторів коефіцієнти нерівності зміняться. Якщо $S(\Lambda)$- матриця переходу, то нові значення векторів $\bar{a},\bar{b}$ обчислюються за формулами $\bar{a}^{(S)} = S\bar{a}S^{-1}, \bar{b}^{(S)} = S\bar{b}S^{-1}$. Але для того, щоб не перевантажувати текст новими позначеннями, будемо використовувати старі позначення. Зауважимо, що матриця виду $\hat{B} \overset{df}{=} \begin{bmatrix} \alpha & \beta \\ -\beta & \alpha \end{bmatrix}$, де $\alpha^2 + \beta^2 = 1$ - матриця повороту вектора 2-вимірного простору на кут φ, певний співвідношеннями $\cos(\varphi) = \alpha, \sin(\varphi) = \beta$. Тому

$$B_1 = r_1 \begin{bmatrix} \cos(\varphi_1) & \sin(\varphi_1) \\ -\sin(\varphi_1) & \cos(\varphi_1) \end{bmatrix},\ r_1 = |\lambda_1| = \sqrt{\alpha^2 + \beta^2}\ .$$

Нерівність (2.30), інваріантність якої розглядається для циклу (2.16) с конкретним початковим значенням $\bar{b}$, означає, що $\forall X \in Orbit(A,\bar{b})(\bar{a},X) \le (\bar{c},\bar{b})$. Алгоритм доведеня інваріантності (2.30) будемо формулювати в еквівалентному вигляді: $\underset{X \in Orbit(A,\bar{b})}{Sup}(\bar{a},X) \le (\bar{c},\bar{b})$.

Розглянемо лінійну форму $a_1x_1 + a_2x_2 + ... + a_nx_n \overset{df}{=} (\bar{a},X)$. Перетворення $X := A * X$ переводить цю форму в (a,AX), а m- кратна ітерація циклу, описувана перетворенням $X := A^m * X$ - переводить в (a,A^mX).

Нехай $X_1 = (x_1,x_2),...,X_k = (x_{2k-1},x_{2k})$, $\bar{a}_1 = (a_1,a_2),...,\bar{a}_k = (a_{2k-1},a_{2k})$. Тоді

$$(\bar{a},X) = (\bar{a}_1,X_1) + ... + (\bar{a}_k,X_k) + a_{2k+1}x_{2k+1} + ... + a_nx_n \qquad (2.31)$$

Перетворення $(\bar{a}, AX)$ лінійної форми може бути записано у вигляді

$$(\bar{a}, AX) = (\bar{a}_1, B_1 X_1) + ... + (\bar{a}_k, B_k X_k) + \lambda_{2k+1} a_{2k+1} x_{2k+1} + ... + \lambda_n a_n x_n \qquad (2.32)$$

а його m - а ітерація – виду

$$(\bar{a}, A^m X) = (\bar{a}_1, B_1^m X_1) + ... + (\bar{a}_k, B_k^m X_k) + \lambda_{2k+1}^m a_{2k+1} x_{2k+1} + ... + \lambda_n^m a_n x_n \qquad (2.33)$$

Перейшовши в (2.33) до представлення у вигляді $B_j = r_j \hat{B}_j$, отримаємо:

$$(\bar{a}, A^m X) = r_1^m (\bar{a}_1, \hat{B}_1^m X_1) + ... + r_k^m (\bar{a}_k, \hat{B}_k^m X_k) + \lambda_{2k+1}^m a_{2k+1} x_{2k+1} + ... + \lambda_n^m a_n x_n .$$

Метод перевірки інваріантності нерівності для діагоналізуємого лінійного оператора. Розглянемо питання про множину значень оператора орбіти $(\bar{a}_1, \hat{B}_1^m X_1) + ... + (\bar{a}_k, \hat{B}_k^m X_k)$ для початкового значення $\bar{b}^{(0)} = (\bar{b}_1^{(0)}, ..., \bar{b}_k^{(0)})$, де $\bar{b}_j = (b_{2j-1}, b_{2j})$, $j = 1, ..., k$. Будемо інтерпретувати пари X_j як точки на 2-мірній площині, а перетворення $\widehat{B}_j \overset{df}{=} \begin{bmatrix} \cos(\varphi_j) & \sin(\varphi_j) \\ -\sin(\varphi_j) & \cos(\varphi_j) \end{bmatrix}$ - як повороти точок X_j на кут φ_j.

Лема 2.6. Нехай

$$B(\varphi) \overset{df}{=} \begin{bmatrix} \cos(\varphi) & \sin(\varphi) \\ -\sin(\varphi) & \cos(\varphi) \end{bmatrix} \qquad (2.34)$$

Якщо кут φ співвимірний з π, орбіта $Orbit(B(\varphi), \bar{b}^{(0)})$ скінченна для будь-якого початкового значення $\bar{b}^{(0)} = (b_1^{(0)}, b_2^{(0)})$, а якщо φ неспівимірний з π, ця орбіта нескінченна і всюди щільна на окружності $x^2 + y^2 = (b_1^{(0)})^2 + (b_2^{(0)})^2$, тобто для будь-якої точки $D(x_0, y_0)$, що лежить на цій окружності, для будь-якого позитивного числа ε в ε - околі точки D існує точка послідовності $\{(x^{(j)}, y^{(j)})\}_{j=0}^{\infty}$.

Доведення є очевидним.

Визначення 2.13. Кути φ, ψ назвемо π-еквівалентними, якщо $\varphi - \psi = r\pi$ для деякого $r \in Rat$.

Нехай B_0, B_1 - оператори виду (2.33) повороту площини на кути φ_0, φ_1 та $Orbit(B_0, X_0), Orbit(B_1, X_1)$ - їх орбіти.

Лема 2.7. Якщо кути φ_0,φ_1 операторів повороту B_0,B_1 не є π-еквівалентними і неспіввимірні з π, для будь-яких точок $D_0(x_0,y_0),D_1(x_1,y_1)$, що лежать на одиничній окружності $x^2+y^2=1$, для будь-якого додатного числа ε в ε- околицях точок D_0,D_1 існують орбіти $Orbit(B_0,X_0),Orbit(B_1,X_1)$ з однаковими номерами.

Доведення. Нехай $\psi_0^{(0)},\psi_1^{(0)}$ - початкові кути орбіт операторів B_0,B_1; ψ_0^*,ψ_1^* - кути точок D_0,D_1. Тоді дія операторів B_0,B_1 визначається формулами

$$\varphi_0^{(k)}=\psi_0+k\varphi_0,\varphi_1^{(k)}=\psi_1+k\varphi_1,k=1,2,\dots$$

Розглянемо різницю $\varphi_1^{(k)}-\varphi_0^{(k)}=(\psi_1-\psi_0)+k(\varphi_1-\varphi_0),k=1,2,\dots$

Позначимо $O(\varepsilon,\varphi)$ ε-околицю кута φ, тобто відкритий інтервал довжини ε з центром в φ. Згідно Лемі 2.6 для будь-якого $\varepsilon>0$ існує таке k_0, що $\varphi_1^{(k_0)}-\varphi_0^{(k_0)}\in O(\varepsilon/2,\psi_1^*-\psi_0^*)$. Далі, існує таке k_1, що $\varphi_0^{(k_0+k_1)}\in O(\varepsilon/2,\varphi_1^{(k_0)}-\varphi_0^{(k_0)})$. Це означає, що знайдеться таке найменше k_2, що $\psi_0^*\in O(\frac{\varepsilon}{2},\varphi_0^{(k_0+k_2k_1)})$. Тоді $\varphi_0^{(k_0+k_2k_1)}\in O(\varepsilon,\psi_0^*),\varphi_1^{(k_0+k_2k_1)}\in O(\varepsilon,\psi_1^*)$.

Лема 2.8. Якщо кути $\varphi_1,\dots,\varphi_k$ операторів повороту $B_1,\dots B_k$ не є попарно π-еквівалентними і несумірні з π, для будь-яких точок $D_1(x_1,y_1),\dots,D_k(x_k,y_k)$, лежать на одиничної окружності $x^2+y^2=1$, для будь-якого позитивного числа ε, в ε- околицях $D_1,\dots,D_k$ існують точки орбіт $Orbit(B_1,X_1),\dots,Orbit(B_k,X_k)$ з однаковими номерами.

При доведенні використовується метод математичної індукції і техніка доведення леми 2.7.

Лема 2.9. Нехай $(\bar a,X)=(\bar a_1,X_1)+\dots+(\bar a_k,X_k)$ - лінійна форма і $B_1,\dots,B_k$ - двовимірні оператори повороту на кути $\varphi_1,\dots,\varphi_k$ відповідно, причому $\varphi_1,\dots,\varphi_k$ не є попарно-еквівалентними і неспіввимірні з π. Визначимо дію оператора A таким чином: $(\bar a,AX)=(\bar a_1,B_1X_1)+\dots+(\bar a_k,B_kX_k)$, а початкове значення $\bar b=(\bar b_1,\dots,\bar b_k)$. Тоді

$$\underset{X \in Orbit(A,\bar{b})}{Sup}(\bar{a},X) = \underset{X \in Orbit(B_1,\bar{b}_1)}{Sup}(\bar{a}_1,X_1) + \cdots + \underset{X \in Orbit(B_k,\bar{b}_k)}{Sup}(\bar{a}_k,X_k) \quad (2.35)$$

Доведення. Для оператора (2.34) розглянемо лінійну форму $(\bar{a},X) = a_1x + a_2y$. За умовою, аргументи цієї форми визначені на окружності O радіуса $r = \sqrt{b_1^2 + b_2^2}$ з центром у початку координат. Форма $(\bar{a},X)$ досягає свого максимуму в точці $M(x_M,y_M)$ дотику прямої $a_1x + a_2y = c$ до цього кола, причому x_M, y_M і з обчислюються методами аналітичної геометрії. Покладемо $d = \sqrt{a_1^2 + a_2^2}$. Неважко обчислити, що

$$x_M = \frac{a_1r}{d}, y_M = \frac{a_2r}{d}, \max = a_1x_M + a_2y_M = rd.$$

Згідно лемі 2.6 у будь-якому ε-околі точки M існують точки орбіти $Orbit(B,\bar{b})$. Тому

$$\underset{X \in Orbit(B_1,\bar{b}_1)}{Sup}(\bar{a}_1,X_1) = \underset{X \in O_1}{max}(\bar{a}_1,X) = rd$$

Нехай $M_1, \ldots, M_k$ - точки максимуму для форм $(\bar{a}_1,X_1), \ldots, (\bar{a}_k,X_k)$ на O_1.

За лемою 2.8 для системи цих точок і будь-якого позитивного ε в ε - околах цих точок існують точки $D_1,..,D_k$ орбіт операторів $B_1,..,B_k$ з початковими значеннями $\bar{b}_1,..,\bar{b}_k$ відповідно. Тому для обчислення $\underset{X \in Orbit(A,\bar{b})}{Sup}(\bar{a},X)$ слід обчислити $\underset{X \in O_1}{max}(\bar{a}_i,X_i)$ і використовувати рівність (2.35).

Лема 2.10. Нехай лінійна форма $(\bar{a},X)$, двовимірні оператори $B_1,\ldots,B_k$, оператор A і початкове значення $\bar{b}$ визначені так само, як і в лемі 2.9. Тоді задачу обчислення $\underset{X \in Orbit(A,\bar{b})}{Sup}(\bar{a},X)$ можна звести до обчислень за формулою (2.24).

Доведення. Припустимо, що $\varphi_1, \ldots, \varphi_l$ - клас еквівалентності φ_1 по відношенню π-еквівалентності $\varphi_i \sim \varphi_j \leftrightarrow \varphi_i - \varphi_j = r_{ij}\pi, r_{ij} \in Rat$. Оскільки точки орбіт всіх операторів B_i лежать на одиничному колі, можна вважати, що $\begin{bmatrix} x_i \\ y_i \end{bmatrix} = R_i \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$, де R_i - оператори повороту на кути $r_{1i}\pi$. Тому $(\bar{a}_1,X_1) +$

$\cdots + (\bar{a}_l, X_l) = (\bar{a}_1 + \bar{a}_2 R_2 + \cdots + \bar{a}_l R_l) X_1$, $a(\bar{a}_1, B_1 X_1) + \cdots + (\bar{a}_l, B_l X_l) = (\bar{a}_1 B_1 + \bar{a}_2 R_2 B_2 + \cdots + \bar{a}_l R_l B_l) X_1$. Таким чином, всі складові лінійної форми з π-еквівалентними кутами можна виразити через одну лінійну форму і в формулі (2.35) можна вважати всі кути попарно нееквівалентними.

Лема 2.11. Нехай $f(x) \in K[x]$ - многочлен і $\lambda_1,...,\lambda_k$ - комплексні корені $f(x)$ такі, що $|\lambda_1| = ... = |\lambda_k| = 1$, $\lambda_j = \cos(\varphi_j) + i\sin(\varphi_j), j = 1...k.$ Тоді проблема обчислення класів π- еквівалентності на множині $\varphi_1,...,\varphi_k$ алгоритмічно розв'язна.

Доведення. Нехай $\varphi, \psi \in \{\varphi_1, \dots, \varphi_k\}, \varphi - \psi = r\pi, r \in Rat, \lambda = \cos(\varphi) + i\sin(\varphi), \mu = \cos(\psi) + i\sin(\psi)$ корінь деякої степені з одиниці. Алгоритми арифметичних дій над алгебраїчними числами викладені, наприклад, в [56]. Основна ідея полягає в наступному: визначимо систему поліноміальних рівностей $(f(u) = 0)\&(f(v) = 0)\&(wv - u = 0)$ і виключимо з неї змінні *u,v* методами теорії виключень [53] або базисів Гребнера [56]. В результаті маємо многочлен *h(w),* один з дільників якого повинен бути мінімальним многочленом деякого кореня з одиниці. Таким чином, $\deg\left(Gcd\left(w^d - 1, h(w)\right)\right) > 0$ для деякого натурального $d \le \deg(h)$.

Лемма 2.12. Нехай f (x) ∈ K[x] - многочлен і $\lambda_1, \dots, \lambda_k$ - його корені. Тоді проблема розпізнавання кількості *k* коренів таких, що $|\lambda_1| = \cdots = |\lambda_k|$, і $|\lambda_1| = \max(|\lambda_1| = \cdots = |\lambda_n|)$ алгоритмічно розв'язна.

Доведення. Покладемо $x = u + iv$. Тоді $f(x)$ можна переписати, виділивши дійсну і уявну частини $f(x)$: $f(u + iv) = re\, f(u, v) + i \cdot imf(u, v)$. Оскільки $|x|^2 = u^2 + v^2$, завдання визначення чисел з рівними модулями можна сформулювати у вигляді системи рівностей

$$(re\, f(u, v) = 0)\&(\, imf(u, v) = 0)\&(w - u^2 - v^2 = 0) \qquad (2.36)$$

Так як змінна v входить в $imf(u, v)$. в непарній степені, то $imf(u, v) = v \cdot imf_1(u, v)$. Поклавши *v* = 0, отримаємо $re\, f(u, 0) = f(u)$. Таким чином, (2.36) спроститься до системи $(f(u) = 0)\&(w - u^2 = 0)$. Іншими словами, для цього

випадку завдання зводиться до відокремлення і обчислення кратності дійсних коренів $f(x)$. При $v \neq 0$ система (2.36) спроститься до $(re\, f(u,v) = 0)\&(\, imf_1(u,v) = 0)\&(w - u^2 - v^2 = 0)$. Оскільки змінна v входить в цю систему в парній степені, при виключенні змінних алгоритм оперує по суті зі змінною $v_1 = v^2$. Виключивши з системи змінні u, v^2 отримаємо поліноміальну рівність $g(w) = 0$, дійсні корені якої - квадрати модулів комплексної частини чисел $\lambda_1, \ldots, \lambda_n$. Тепер завдання зводиться до відокремлення і визначення кратності коренів $g\ (w)\ = 0$. Якщо λ_i - максимальний дійсний корінь $f(x)$, а λ_j- максимальний дійсний корінь *g (x)* і $\lambda_i = \lambda_j$ то $\deg\big(Gcd(f,g)\big) > 0$ і λ_i - дійсний корінь $Gcd(f,g)$). Нехай p - кратність λ_i, а q- кратність λ_j і $\lambda_i = \lambda_j$.Тоді кратність λ_j дорівнює $p + 2q$. Якщо, $\lambda_i \neq \lambda_j$, кратність шуканого максимального кореня дорівнює або p (випадок 1), або $2q$ (випадок 2).

Приклад 2.10. Алгоритми алгебраїчних обчислень задачі доказу інваріантності нерівності.

Нехай $A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 2 & 0 & 0 \end{bmatrix}$.

1. Обчислення характеристичного многочлена:

$$f(x) = |A - xE| = \begin{bmatrix} -x & 1 & 0 \\ 0 & -x & 1 \\ 2 & 0 & -x \end{bmatrix} = -x^3 + 2.\ \ f(x) = x^3 - 2.$$

$\lambda_1 = \sqrt[3]{2},\ \lambda_2 = \sqrt[3]{2} * \varepsilon,\ \lambda_3 = \sqrt[3]{2} * \varepsilon^2,\ \ \varepsilon = \cos(\frac{2\pi}{3}) + i * \sin(\frac{2\pi}{3}).\ \varepsilon$ - первісний корінь степені 3 з одиниці.

2. Алгоритм леми 2.12. Система рівнянь леми:

$$f(u + iv) - re\, f(u,v) + i * im\, f(u,v);$$

$$(ref(u,v) - 0)\&(imf(u,v) - 0)\&(w - u^2 - v^2 - 0);$$

$$x^3 - 2 = 0 \to (u + iv)^3 - 2 = 0;$$

$$\begin{cases}(u^3-3uv^2-2)=0\\ 3u^2v-v^3=0\end{cases}$$

$$\rightarrow \begin{cases}u^3-3uv^2-2=0\\ 3u^2-v^2=0\end{cases} \text{ (випадок 1) v } \begin{cases}u^3-2=0\\ v=0\end{cases} \text{ (випадок 2).}$$

Випадок 1:

$$\begin{cases}u^3-3uv^2-2=0\\ 3u^2-v^2=0\\ w=u^2+v^2\end{cases} \rightarrow \begin{cases}v^2=\dfrac{u^3-2}{3u}\\ 3u^2-v^2=0\\ w=u^2+v^2\end{cases} \rightarrow \begin{cases}uv^2+\dfrac{3}{4}=0\\ 4u^3+1=0\\ w^3-4=0\end{cases}.$$

Отже, q = 1, p = 1. Многочлен f (x) = x^3 -2 має p + 2q = 3 корені з максимальним модулем.

3. Алгоритм леми 2.11.

3.1. Перевірка співрозмірності кута φ з π. Маємо систему

$$(f(x)=0)\&(f(w^2)=0)\&(x-uw=0),$$

$$\begin{cases}x^3-2=0\\ w^6-4=0\\ x-uw=0\end{cases} \rightarrow \begin{cases}x^3-2=0\\ w^3-2=0\\ x-uw=0\end{cases} \rightarrow \begin{cases}(uw)^3-2=0\\ w^3-2=0\\ x-uw=0\end{cases} \rightarrow \begin{cases}u^3-1=0\\ w^3-2=0\\ -x-uw=0\end{cases} \rightarrow$$

$$\rightarrow \text{Gcd}(u^3-1, u^2+u+1)=u^2+u+1 \rightarrow \varphi=2\pi/3,$$

тобто φ співрозмірний з π.

3.2. Перевірка співрозмірності різниці кутів $\varphi_u-\varphi_v$ з π. Система має вигляд $(f(u)=0)\&(f(v)=0)\&(wv-u=0)$,

$$\begin{cases}u^3-2=0\\ v^3-2=0\\ wv-u=0\end{cases} \rightarrow \begin{cases}w^3v^3-2=0\\ v^3-2=0\\ wv-u=0\end{cases} \rightarrow \begin{cases}w^3-1=0\\ v^3-2=0\\ wv-u=0\end{cases} \rightarrow$$

$$\rightarrow \text{Gcd}(w^3-1, w^2+w+1)=w^2+w+1 \rightarrow \varphi_u-\varphi_v=2\pi/3,$$

тобто кути φ_u, φ_v π-еквівалентні.

3.3. Подання AX для лінійної форми виду (2.32):

$$A\begin{bmatrix}x\\ y\\ z\end{bmatrix}=(\sqrt[3]{2}\begin{bmatrix}\cos\left(\frac{2\pi}{3}\right) & \sin\left(\frac{2\pi}{3}\right)\\ -\sin\left(\frac{2\pi}{3}\right) & \cos\left(\frac{2\pi}{3}\right)\end{bmatrix}\begin{bmatrix}x\\ y\end{bmatrix}, \sqrt[3]{2z})^{\text{T.}}$$

3.4. Обчислення матриці переходу до жорданової форми.

- Обчислення власних векторів матриці A = $\begin{pmatrix}0 & 1 & 0\\ 0 & 0 & 1\\ 2 & 0 & 0\end{pmatrix}$.

Обчислити систему однорідних лінійних рівнянь $(A-\lambda E)(x,y,z)^T=0$, причому обчислення здійснюються в полі $K(\lambda)$ по модулю λ^3-2.

Ранг системи $\begin{cases}\lambda x-2z=0\\ x-\lambda y=0\\ y-\lambda z=0\end{cases}$ дорівнює двом. Фундаментальний розв'язок системи є вектор $s=(\lambda^2,\lambda,1)$. Власні вектори оператора A:

$$s_1=(\lambda^2{}_1,\lambda_1,1), s_2=(\lambda^2{}_2,\lambda_2,1), s_3=(\lambda^2{}_3,\lambda_3,1).$$

- Обчислення матриці переходу S і оберненої матриці S^{-1}:

$$S=\begin{bmatrix}\lambda_1^2 & \lambda_2^2 & \lambda_3^2\\ \lambda_1 & \lambda_2 & \lambda_3\\ 1 & 1 & 1\end{bmatrix}, S^{-1}=\frac{1}{\det(S)}\begin{bmatrix}\lambda_2-\lambda_3 & \lambda_2^2-\lambda_3^2 & \lambda_2^2\lambda_3-\lambda_2\lambda_3^2\\ \lambda_1-\lambda_3 & \lambda_1^2-\lambda_3^2 & \lambda_1^2\lambda_3-\lambda_1\lambda_3^2\\ \lambda_1-\lambda_2 & \lambda_1^2-\lambda_2^2 & \lambda_1^2\lambda_2-\lambda_1\lambda_2^2\end{bmatrix}.$$

Теорема 2.16. Проблема доказу інваріантності нерівності $L(\bar{a},\bar{c},X,\bar{b})$ для циклу (2.15) з діагоналізуємим лінійним оператором A і з даною початковою точкою $\bar{b}$ алгоритмічно розв'язна.

Доведення. Оскільки оператор A діагоналізуємий, представлення $(\bar{a},A^mX)$ має вигляд

$$(\bar{a},A^mX)=r_1^m\left(\overline{a_1},\widehat{B_1^m}X_1\right)+\cdots+r_k^m\left(\overline{a_k},\widehat{B_k^m}X_k\right)+\lambda_{2k+1}^m a_{2k+1}x_{2k+1}+\cdots+\lambda_n^m a_n x_n. \quad (2.36)$$

Нехай p і q - числа, визначені в лемі 2.12 для характеристичного полінома *f(x)* оператора *A*. Для визначеності будемо вважати, що власні числа оператора A з максимальними модулями мають початкові номери $\lambda_1,\lambda_2,\dots,\lambda_{2q-1},\lambda_{2q},\lambda_{2q+1},\dots,\lambda_{2q+p}$, причому $\lambda_1=\overline{\lambda_2},\dots,\lambda_{2q-1}=\overline{\lambda_{2q}}$. Тоді (2.36) можна представити у вигляді

$$(\bar{a},A^mX)=r_1^m\left(\overline{a_1},\widehat{B_1^m}X_1\right)+\cdots+r_k^m\left(\overline{a_q},\widehat{B_q^m}X_q\right)+\lambda_{2q+1}^m a_{2q+1}x_{2q+1}+\cdots$$
$$+\lambda_{2q+p}^m a_{2q+p}x_{2q+p}+L_1^{(m)},$$

де $L_1^{(m)}-m$-та ітерація лінійної форми виду (2.36) від решти змінних. Нерівність (2.30) можна переписати у вигляді

$$\left(\overline{a_1},\widehat{B_1^m}X_1\right)+\cdots+\left(\overline{a_q},\widehat{B_q^m}X_q\right)+\lambda_{2q+1}^m a_{2q+1}x_{2q+1}+\cdots+\varepsilon_{2q+p}^m a_{2q+p}x_{2q+p}$$
$$+\frac{1}{r_1^m}L_1^{(m)}\le\frac{1}{r_1^m}\left(\bar{c},\bar{b}\right).$$

Позначимо $L_0^{(m)}$ лінійну форму

$$\left(\overline{a_1}, \widehat{B_1^m} X_1\right) + \cdots + \left(\overline{a_q}, \widehat{B_q^m} X_q\right) + \varepsilon_{2q+1}^m a_{2q+1} x_{2q+1} + \cdots + \varepsilon_{2q+p}^m a_{2q+p} x_{2q+p},$$

де $\varepsilon_j = \pm 1$. Отримаємо

$$L_0^{(m)} + \frac{1}{r_1^m} L_1^{(m)} \leq \frac{1}{r_1^m}\left(\bar{c}, \bar{b}\right). \qquad (2.37)$$

Так як $r_1 > \max\left(\left|\lambda_{2p+q+1}\right|, \ldots, |\lambda_n|\right)$, при $m \to \infty$ маємо $\frac{1}{r_1^m} L_1 \to 0$. Отже для будь-якого $\delta > 0$ існує таке натуральне число m_0, що при $m > m_0$ маємо $\left|\frac{1}{r_1^m} L_1^{(m)}\right| < \delta$.

Мають місце такі випадки.

1. Якщо $r_1 < 1$, то $\lim_{m \to \infty} \frac{1}{m}\left(\bar{c}, \bar{b}\right) = \infty$. Тоді:

- При $\left(\bar{c}, \bar{b}\right) < 0$ для будь-якого $\varepsilon > 0$ існує таке $m_1 < m$, при якому

$$L_0^{(m)} + \frac{1}{r_1^m} L_1^{(m)} < L_0^{(m)} + \delta < -\varepsilon,$$

тобто при $m > max\ (m_0, m_1)$ нерівність (2.30) не виконується;

- При $\left(\bar{c}, \bar{b}\right) < 0$ маємо $L_0^{(m)} + \frac{1}{r_1^m} L_1^{(m)} < L_0^{(m)} + \delta < +\infty$, тобто при $m > max(m_0, m_1)$ нерівність (2.30) виконується.

2. Якщо $r_1 = 1$, то $\lim_{m \to \infty} \frac{1}{m}\left(\bar{c}, \bar{b}\right) = \left(\bar{c}, \bar{b}\right)$. При $m_1 > m_0$ маємо $L_0^{(m)} + \frac{1}{r_1^m} \cdot L_1^{(m)} < L_0^{(m)} + \delta < \left(\bar{c}, \bar{b}\right)$. Отже, для виконання нерівності (2.30) при $m > max(m_0, m_1)$ повинна виконуватися умова $L_0^{(m)} < \left(\bar{c}, \bar{b}\right)$.

3. Якщо $r_1 > 1$, то $\lim_{m \to \infty} \frac{1}{m}\left(\bar{c}, \bar{b}\right) = 0$. При $m_1 > m_0$ маємо $L_0^{(m)} + \frac{1}{r_1^m} \cdot L_1^{(m)} < L_0^{(m)} + \delta < 0$. Отже, для виконання нерівності (2.30) при $m > max(m_0, m_1)$ має виконуватися умова $L_0^{(m)} < 0$.

Нерівність виду $L_0^{(m)} < c$ для довільного c на довільній множині S еквівалентна $\underset{X \in S}{Sup} L_0^{(m)} < c$. Тому незалежно від випадків 1-3, доведення інваріантності зводиться до обчислення

$$\underset{X \in Orbit(A, \bar{b})}{Sup} L_0^{(m)},$$

$$L_0^{(m)} = \left(\bar{a}_1, \hat{B}_1^m \bar{b}_1\right) + \cdots + \left(\bar{a}_q, \hat{B}_q^m \bar{b}_q\right) + \varepsilon_{2q+1}^m a_{2q+1} b_{2q+1} + \cdots + \varepsilon_{2q+p}^m a_{2q+p} b_{2q+p}.$$

Алгоритм обчислення $\underset{X \in Orbit(A,\bar{b})}{Sup}\left(\bar{a}_1, \hat{B}_1^m \bar{b}_1\right) + \cdots + \left(\bar{a}_q, \hat{B}_q^m \bar{b}_q\right)$ наведено в лемах 2.9, 2.11. Обчислення лінійної частини $\underset{X \in Orbit(A,\bar{b})}{Sup}\varepsilon_{2q+1}^m a_{2q+1} b_{2q+1} + \cdots + \varepsilon_{2q+p}^m a_{2q+p} b_{2q+p}$ здійснюється безпосередньо для парного і непарного значень m.

Загальний алгоритм доведення інваріантності нерівності (2.30) для циклу (2.13) і даного початкового значення $\bar{b}$ полягає в наступному.

Крок 1. Привести нерівність до виду (2.37).

Крок 2. Обчислити значення $m^* = max(m_0, m_1)$ і визначити приналежність до одного з варіантів: 1, 2 або 3.

Крок 3. Перевірити виконання умови інваріантності для випадку, визначеного в п. 2, і для значень $m > m^*$.

Крок 4. Перевірити нерівність (2.30), виконуючи цикл (2.15) для всіх значень $m \le m^*$.

Теорема 2.17. Проблема доведення інваріантності нерівності $L(\bar{a}, \bar{c}, X, \bar{b})$ для циклу (2.15) (тобто з даною передумовою $S(b)$) алгоритмічно розв'язна.

Доведення. Лінійну півалгебраїчну множину $S = \{\bar{b} \mid S(\bar{b})\}$ можна представити у вигляді об'єднання опуклих багатогранних множин, оскільки *S(b)*можна представити у вигляді $S_1(b) \vee \ldots \vee S_l(b)$ - систем лінійних нерівностей. В лемах 2.2, 2.3 доведено, що виконання лінійної нерівності для будь-якої точки випуклого багатогранника випливає з його виконання на всіх вершинах цього багатогранника. Тому доведення інваріантності $L(\bar{a}, \bar{c}, A^m X, \bar{b})$ на S(b) полягає в його перевірці для всіх *S1(b),… ,Sl(b).*

Теорема 2.18. Проблема завершуваності циклу (2.15) алгоритмічно розв'язна.

Доведення. Цикл (2.15) розходиться тоді і тільки тоді, коли умова *U(x, b)* - інваріант циклу (2.16).

Висновки до розділу 2

У підрозділі 2.1. приведено поняття власного полінома лінійного оператора та алгоритм побудови власних поліномів, визначено зв’язок між власними поліномами лінійних операторів та поліноміальними інваріантами лінійних циклів програм. Основні алгоритми задачі доказу та генерації поліноміальнихінваріантних рівностей - побудова множини L – інваріантів циклів для операторів, у жордановій формі яких є нетривіальні блоки; теорема 2.12, в якій встановлено структуру базису Гребнера ідеалу поліноміальних інваріантів лінійного циклу з довільним невироджених лінійним оператором в тілі циклу і теорема 2.14 про алгоритмічну розвязуваність проблеми побудови базису ідеалу інваріантів «діагоналізуємої частини» лінійного оператора в разі неприводимості його мінімального характеристичного полінома. У зв'язку з цим, за теоремою 2.12, інваріанти лінійного оператора можна класифікувати як внутрішньоклітинкові - ті, які притаманні кожній жорданової клітинці лінійного оператора, і міжклітинкові - ті, які притаманні його діагоналізуємій частині. Внутрішньоклітинкові інваріанти обчислюються безпосередньо за формулами $x_j = \frac{z}{c}(a_j + \frac{C_1(\lambda\frac{cy-bz}{cz})}{\lambda}a_{j+1} + ... + \frac{C_{n-j}(\lambda\frac{cy-bz}{cz})}{\lambda^{n-j}}a_n)$. Існування міжклітинкових інваріантів залежить від існування нетривіальних мультиплікативних співвідношень між власними числами лінійного оператора (теорема 2.2).

Для лінійних операторів з непривідним мінімальним характеристичним многочленом проблема побудови базису множини мультиплікативних співвідношень між його власними числами алгоритмічно розв'язна, але алгоритм теореми 2.14 неефективний зважаючи на дуже великі степені многочлена $S(x)$, який потрібно розкладати на множники.

У підрозділі 2.2. представлено нові методи доказу інваріантності системи лінійних нерівностей і завершуваності лінійно визначених ітеративних циклів імперативних програм. Тілом циклу є лінійний оператор, який перетворює вектор змінних програми. У методі враховується передумова циклу та умова його повторення у вигляді сукупності системи лінійних нерівностей. В основі методу лежить побудова та аналіз спектру лінійного оператора тіла циклу та обчислення числа його ітерацій, після виконання яких інваріантність або підтверджується або спростовується.

Запропоновані методи може бути покладено в основу загального алгоритму доказу інваріантності системи лінійних нерівностей та поліноміальних ріностей для лінійно-визначених програм.

Таким чином, відповідно до завдань дисертаційного дослідження у даному розділі:

1. Розглянуто проблему генерації поліноміальних інваріантів ітераційних циклів з оператором ініціалізації циклу і невиродженим лінійним оператором в тілі циклу.

2. Приведено алгоритми обчислення основних інваріантів лінійного оператора жорданової клітинки та основних інваріантів діагоаналізуємого лінійного оператора з непривідним мінімальним характеристичним поліномом.

3. Представлено новий метод доказу інваріантності системи лінійних нерівностей і завершення деяких лінійних ітераційних циклів імперативних програм, дані яких є елементами конструктивного лінійно впорядкованого поля.

Результати дослідження апробовано на Міжнародній науковій конференції ICTERI, м. Львів, травень 2015 р [57] та Міжнародній науковій конференції Сучасна інформатика: проблеми, досягнення та перспективи розвитку, м. Київ, 13-15 грудня, 2017 р. (доповідь за темою - Loop invariants as a key problem of programs static analysis (Інваріанти циклів як ключова проблема статичного аналізу програм)).

Список літератури до Розділу 2

1. Floyd R. W. Assigning Meanings to Programs / R. W. Floyd // Proceedings of Symposium on Applied Mathematics, Vol. 19, J.T. Schwartz (Ed.), American Mathematical Society, Providence. – 1967. - P. 19-32.

2. Hoare C. A. R. An axiomatic basis for computer programming / C. A. R. Hoare //Communications of the ACM. – 1969. – Т. 12. – №.10. – С. 576-580.

3. Cocke J., Schwartz J.T. Programming Language and their Compilers / J. Cocke, J.T. Schwartz // Courant Institute of Mathem. Sciences. – New York University. – 1970. – N.Y. – P. 3–21.

4. Allen F.E. Interprocedural analysis / F. E. Allen // ACM SIGPLAN Notices. – 1976. – Vol. 6, N 7. – P. 23 – 31.

5. Летичевский А.А. Эквивалентность и оптимизация программ / А.А. Летичевский // Труды междунар. симпозиума по теоретическому программированию. – Новосибирск, 1972. – С. 166–180.

6. Kildall G.A. A unified approach to program optimization / G.A. Kildall // Conf. Rec. of ACM Symp. on Prince. of Program Languages, Boston, Massachusetts, Oktober 1–3, 1973. – P. 194–206.

7. Caplain M. Finding Invariant Assertions for Proving Programs / M. Caplain // Proc. of the intern. Conf. on Reliable Software, Los Angeles, California, April 21 – 23 1975. ACM: New York, NY, USA - 1975.- pp. 165-171.

8. Kam J.B., Ullman D.J. Monotone data flow analysis frame works / J.B. Kam, D.J. Ullman // Acta Inform. – 1978. – Vol. 3. – P. 305–318.

9. German S., Wegbreit B. A synthesizer of inductive assertions / S. German, B. Wegbreit // IEEE Transactions on Software Engineering. – 1975. - №1(1). – P. 68–75.

10. B. Wegbreit. The Synthesis of Loop Predicates / B. Wegbreit // Communications of the ACM. – 1974. - №17(2). - P. 102–112.

11. B. Wegbreit. Property extraction in well-founded property sets / B. Wegbreit // IEEE Transactions on Software Engineering. – 1975. - №1(3). – P. 270–285.

12. Research in Interactive ProgramProving Techniques. Technical report / B.Elspas, M. Green, K. Levitt, R. Waldinger. – Menlo Park, California, USA: Stanford Research Institute, 1972.

13. Katz S., Manna Z. Logical Analysis of Programs / S. Katz, Z. Manna // Communications of the ACM. – 1976. - №19(4). – P. 188–206.

14. Letichevsky A.A. About one approach to program analysis / A.A. Letichevsky // Cybernetics. - 1979. - № 6. - P. 1-8.

15. Iterative methods of program analysis / A.B. Godlevsky, Y.V. Kapitonova, S.L. Krivoy, A.A. Letichevsky // Cybernetics. – 1989. - №2.- P.9-19.

16. Кривой С. Л. О поиске инвариантных соотношений в программах / С. Л. Кривой // Математическая теория проектирования вычислительных машин. – Киев: Изд. Ин-та кибернетики АН УССР, 1978. – С. 35–51.

17. Sabelfeld V. K. Equivalente Transformationen f¨ur Flussdiagramme / V. K Sabelfeld // Acta Informatica. – 1978. – №10. – P. 127–155.

18. Кривой С. Л. Об одном алгоритме поиска инвариантных соотношений в программах / С. Л. Кривой // Кибернетика. – 1981. – № 5. – С. 12–18.

19. Lvov M. S. About one algorithm of program polynomial invariants generation / M. S. Lvov // Proc. Workshop on Invariant Generation: (Techn. rep.) Univ. of Linz / Eds. M. Giese, T. Jebelean. No 0707. – (RISC Report Series). Linz (Austria), 2007. – P. 85–99.

20. Львов М. С., Крекнін В. А. Нелінійні інваріанти лінійних циклів та власні поліноми лінійних операторів / М. С. Львов, В. А. Крекнін // Кибернетика и системный анализ. – 2012. – № 2. – С. 126–140.

21. Сабельфельд В.К. Учет свойств операций при глобальном анализе свойств программ / В.К. Сабельфельд // Математическая теория программирования. – Новосибирск: ВЦ СО АН СССР. – 1985. – С. 132–149.

22. Иткин В.Э. Логико–термальная эквивалентность схем программ / В.Э. Иткин // Кибернетика. – 1972. – № 1. – С. 5–27.

23. Cousot P., Halbwochs N. Discovery of Linear Restraints Among Variables of Program / P. Cousot, N. Halbwochs // Conf. Record of the 5–th Annual ACM Symposium on Princieples of Programming Languages. – USA. – 1978. – P. 84–96

24. Cousot P., Halbwochs N. Automatic Discovery of Linear Restraints among Variables of a Program. / P. Cousot, N. Halbwochs // Conference Record of the Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Tucson, Arizona, ACM Press, New York, NY. - 1978. – P. 84–97.

25. Karr M. Affine relationships among variables of a program / M. Karr // Acta Informatica. – 1976. - №6. – P. 133–151.

26. Cousot P., Cousot R. Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints / P. Cousot, R. Cousot // In Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Los Angeles, California. - ACM Press, New York, NY. - 1977. – P. 238–252.

27. Львов М.С. О вычислении инвариантов программ, интерпретированных над областью целостности / М.С. Львов // Кибернетика. – 1984. – №2. – С.23-28.

28. Львов М.С. Об инвариантных неравенствах для состояний схем программ, интерпретированных над векторным пространством / М.С. Львов // Кибернетика. – 1985. – №2.– С.111-112

29. . Львов М.С. Инвариантные неравенства в программах, интерпретированных над упорядоченными полями / М.С. Львов // Кибернетика. – 1986. – №5. – С.22-27

30. Львов М.С. Инвариантные равенства малых степеней в программах, определенных над полем/ М.С. Львов // Кибернетика. – 1988. – №1 . – С. 108-110

31. Letichevsky A., Lvov M. Discovery of Invariant Equalities in Programs over Data Fields / A. Letichevsky, M. Lvov // Applicable Algebra in Engineering, Communication and Computing. – 1993. – №4. – P. 21-29.

32. Müller-Olm M., Seidl H. Precise interprocedural analysis through linear algebra / M. Müller-Olm, H. Seidl // Proc. of Symposium on Principles of Programming Languages. - Venice, Italy, January 14-16, 2004. ACM: New York, NY, USA, 2004. - P. 330-341.

33. Müller-Olm M., Seidl H. Computing polynomial program invariants / M. Müller-Olm, H. Seidl // Inf. Process. Lett. Elsevier North-Holland, Inc. Amsterdam. - 2004.- № 91(5). - P. 233-244.

34. Kapur D. Automatically Generating Loop Invariants Using Quantifier Elimination–Preliminary Report / D. Kapur // IMACS Intl. Conf. on Applications of Computer Algebra. – 2004.

35. Sankaranarayanan S. Non-linear loop invariant generation using Gröbner bases / S. Sankaranarayanan, H. Sipma, Z. Manna // Proc. of Symposium on Principles of Programming Languages. - Venice, Italy, January 14-16, 2004. ACM: New York, NY, USA. - 2004.- P. 318-329.

36. Rodríguez-Carbonell E., Kapur D. Automatic generation of polynomial loop invariants: algebraic foundations / E. Rodríguez-Carbonell, D. Kapur // Proc. Of International Symposium on Symbolic and Algebraic Computation.- Santander, Spain, July 4-7, 2004.- ACM: New York, NY, USA 2004.- P. 266-273.

37. Rodríguez-Carbonell E., Kapur D. Automatic generation of polynomial invariants of bounded degree using abstract interpretation / E. Rodríguez-Carbonell, D. Kapur // Sci. Comput. Program..- Elsevier North-Holland, Inc. Amsterdam, The Netherlands.- 2007. - №64(1). - P.54-75.

38. Kovács L. I., Jebelean T. An Algorithm for Automated Generation of Invariants for Loops with Conditionals / L. I. Kovács, T. Jebelean // Proc. of Intern. Symposium on Symbolic and Numeric Algorithms for Scientific

Computing . – Timisoara, Romania, 25-29 Sept. 2005. IEEE Computer Society: 2005.- P. 245-249.

39. Львов М.С. Полиномиальные инварианты линейных циклов / М.С.Львов // Кибернетика и системный анализ. –2010 - № 4. – С. 159–168.

40. Львов М.С., Крекнин В.А. Собственные полиномы линейных операторов и полиномиальные инварианты линейных циклов программ. / В.А. Крекнин, М.С.Львов // Науковий часопис національного педагогічного університету ім. М.П.Драгоманова. – 2010. – № 11. – С. 150–169.

41. Робинсон А. Введение в теорию моделей и метаматематику алгебры / .А. Робинсон. - М.:"Наука", ГЗФМЛ, 1967. – 376 с.

42. Кривой С.Л., Ракша С.Г. Поиск инвариантных линейных зависимостей в программах / С.Л. Кривой, С.Г. Ракша // Кибернетика. –1984. – №6. – С. 23–28.

43. Итеративные методы анализа программ. Равенства и неравенства / А.Б. Годлевский, Ю.В. Капитонова, С.Л. Кривой, А.А. Летичевский // Кибернетика. - 1990. - №3. - С. 1-10.

44. Львов М.С. Метод доказательства инвариантности линейных неравенств для линейных циклов / М.С. Львов // Кибернетика и системный анализ. – 2014. – № 4. – С. 80-85.

45. Годлевский А.Б., Кривой С.Л. Применение техники смешанных вычислений к построению эффективных алгоритмов унификации и приведения выражений / А.Б. Годлевский, С.Л. Кривой // Тезисы докл. Всесоюз. конф. "Методы трансляции и конструирования программ". – Новосибирск, 1988. – С. 59–62.

46. Hoare T. The verifying compiler: A Grand challenge for computing research / T. Hoare // J. of the ACM. – 2003. – № 50(1) – P. 63–69.

47. Invariant semidefinite programs / C. Bachoc, D. C. Gijswijt, A. Schrijver, F. Vallentin // Handbook on semidefinite, conic and polynomial optimization. – Springer US, 2012. – P. 219-269.

48. Müller-Olm M. Interprocedurally analyzing polynomial identities / M. Müller-Olm, M. Petter, H. Seidl // STACS 2006. – Springer Berlin Heidelberg, 2006. – С. 50-67.

49. Rodríguez-Carbonell E., Kapur D. Generating all polynomial invariants in simple loops / E. Rodríguez-Carbonell, D. Kapur //Journal of Symbolic Computation. – 2007. – №. 42(4). – P. 443-476.

50. Sipma H. B., Manna Z. Non-linear loop invariant generation using Gröbner bases / H. B. Sipma, Z. Manna //ACM SIGPLAN Notices. – 2004. – №. 39(1). – P. 318-329.

51. Furia C. A.. A survey of loop invariants / Furia C. A., Meyer B., Velder S. // CoRR. – 2012.

52. Ван-дер-Варден Б. Алгебра. Изд. 2-ое.` / Б. Ван-дер-Варден. – М.: ГРФМЛ, 1979. – 624 с.

53. Ходж В., Пидо Д. Методы алгебраической геометрии. Том 1. / В. Ходж, Д. Пидо. - М.: ИЛ, 1954. - 462 с.

54. Геометрическая теория инвариантов. / Дьедонне Ж., Керрол Дж, Мамфорд Д. - М.:Мир, 1974.- 280 с.

55. Львов М.С. О структуре полиномиальных инвариантов линейных циклов / М.С. Львов // Кибернетика и системный анализ. — 2015. — № 51(3) - С. 143-156.

56. Компьютерная алгебра: символьные и алгебраические вычисления / Под ред. Б.Бухбергера, Дж. Коллинза, Р.Лооса. – М.:Мир, 1986. - 392 с.

57. Lvov M. The Static Analysis of Linear Loops [Електронний ресурс] / M. Lvov, Y. Tarasich // CEUR Workshop Proceedings. – 2015. – Режим доступу до ресурсу: http://ceur-ws.org/Vol-1356/paper_79.pdf.

РОЗДІЛ 3. КАНОНІЧНІ ФОРМИ ЛІНІЙНИХ НАПІВАЛГЕБРАЇЧНИХ ФОРМУЛ НАД ТИПАМИ ДАНИХ ТА ЇХ ВИКОРИСТАННЯ В ЗАДАЧАХ ВЕРИФІКАЦІЇ ПРОГРАМ.

3.1. Інструменти побудови доказів. Спрощувачі

Інструментів побудови доказів достатньо багато. Наприклад, - ACL2, Otter, j'Imp, Metis, MetiTarski, Jape, PVS, Leo II, EQP, SAD, PhoX, KeYmaera, Gandalf, Tau, E, SNARK, Vampire, Waldmeister, Saturate, Theorem Proving System(TPS) та ін.

Усі їх можна розділити на декілька типів:

– *Інструменти, засновані на розширеннях пропозиціональної логіки або логік першого порядку.*

Для пропозиціональної логіки існують алгоритми, що перевіряють правильність тверджень, але це NP-повна задача. Для обчислення першого порядку множина доказових тверджень є нерозв'язною, але перелічуваною - можна за допомогою алгоритму поступово побудувати всі доказові твердження, але не можна перевірити, що твердження неможливо довести. Щоб отримати достатню для дедуктивного аналізу програм виразність часто використовують розширення логіки першого порядку різними простими теоріями, а також індуктивними правилами виведення, як явними, так і неявними, представленими у вигляді правил переписування умов (term rewriting rules). У цій категорії найбільш відомі наступні інструменти: ACL2, E (та E-SETHEO), KeY, Vampire, Waldmeister, Darwin.

ACL2 логіка та мова програмування, засобами якої ви можете моделювати комп’ютерні системи. Має інструмент для доведення властивостей цих моделей. «ACL2» означає «Обчислювальна логіка для Applicative Common Lisp».

ACL2 є частиною сімейства докзувачів Бойєр-Мура автори якого автори 2005 ACM Software System Award [1, 2]. Знаходиться у вільному доступі

відповідно до умов ліцензійного файлу з наданого ACL2. Ліцензію, авторське право та іншу інформацію можна знайти в документації ACL2.

E (і його відгалуження E-SETHEO [3]).

E – високопродуктивний доказувач теорем для логіки першого порядку з рівностями.Базується на еквaціональних суперпозиційних обчисленнях і використовує чисто еквaціональну парадигму. Був інтегрований в інші прувери, та став одним з кращих систем в декількох конкурсах доведення теорем. E розроблений Stephan Schulz (Automated Reasoning Group, TU Munich). E та SETHEO були об'єднані (з іншими системами) в комбінований прувер E-SETHEO [4, 5].

KeY. Система KeY є формальним інструментом розробки програмного забезпечення, яке прагне інтегрувати проєкт ування, впровадження, формальні специфікації та формальну верифікацію об'єктно-орієнтованого програмного забезпечення, настільки легко, наскільки це є можливим. В основі системи є прувер для доведення теорем першого порядку Dynamic Logic для Java зі зручним графічним інтерфейсом [6, 7].

Проєкт було розпочато в листопаді 1998 року в Університеті Карлсруе. В даний час розроблення системи є спільним проєкт ом Технологічного інституту Карлсруе та Технологічного університет уЧалмерса (Gothenburg and TU Darmstadt). Система доступна для завантаження.

Vampire. Vampire автоматичний доказувач теорем для класичної логіки першого порядку, розроблений у Школі комп'ютерних наук в Університеті Манчестера Андрієм Воронковим разом з Kryštof Hoder і раніше з Олександром Рязановим. Є переможцем «Кубок світу для доказувачів теорем» (CADE ATP (Automated Theorem Proving) System Competition (CASC)) в найпрестижнішій CNF (MIX) одинадцять разів (1999, 2001-2010) [8].

Waldmeister. Waldmeister – доказувач теорем для еквaціональної логіки першого порядку. Базується на Knuth-Bendix завершені використовуваному як доведення процедур. Основною перевагою є те, що ефективність була досягнута з точки зору часу, а також простору [9].

Darwin. Darwin являє собою автоматизовану програму автоматичного доведення теореми для логіки першого порядку. В реалізовановаріант обчислення не вбудовані рівності, він автоматично аксіоматизує дані проблеми. Darwin -це перша реалізація моделі еволюційних обчислень. Поточна версія Darwin реалізує варіанти першого порядку розповсюдження правил блоку логічного виведення за аналогією в обмеженому вигляді резолюції блоку і поглинання на одиницю пропозиції. Щоб зберегти повноту, вона включає в себе версії першого порядку (двійкових) пропозиціональних правил поділу виведення [10].

– *Інструменти, засновані на логіках вищих порядків*.

Вони завжди інтерактивні, оскільки теореми в логіках вищих порядків не є навіть перелічуваними. Проведення доказування з їх допомогою часто потребує втручання людини (для формулювання потрібної допоміжно леми або змінити стратегії доведення). Ці інструменти можуть відрізнятися один від одного наявністю та обсягом допоміжних теорій, доступних для використання. До найбільш відомих та широковикористовуваних систем належать: PVS, HOL, Isabelle, Coq.

PVS [11].P VS – система верифікації: специфічна мова інтегрована з інструментами підтримки і доказувачем теорем. Систему призначено для впровадження інноваційного підходу до механізації формальних методів і є досить потужним для використання для важливих додатків. PVS є дослідницьким прототипом: прувер розвивається і поліпшується у міру розвитку або необхідності застосування нових можливостей.

HOL [12] (є розвитком системи LCF). HOL - інтерактивна програма автоматичного доведення теорем, є помічником доведення для логіки високого порядку: середовище програмування, в якому можуть бути доведені теореми та реалізовані інструменти доказу. Вбудовані процедури прийняття рішень і програми для доведення теорем можуть автоматично створити безліч простих теорем (користувачу, можливо, доведеться доводити важкі теореми самому). Механізм Oracle забезпечує доступ до зовнішніх програм, таких як SMT і системи

BDD. HOL особливо підходить в якості платформи для реалізації поєднання дедукції, виконання і перевіркивластивостей.

Isabelle [13]. Isabelle -це універсальний доказувач. Математичні формули повинні бути виражені на формальній мові. Надає інструменти для доказу цих формул в логічному обчисленні. Isabelle був спочатку розроблена в Кембриджському університеті і технічному університеті Мюнхена, але тепер включає в себе численні внески від організацій і приватних осіб по всьому світу. Найпоширеніший примірник в даний час Isabelle / HOL, який забезпечує більш високий порядок логіки доказу теорем середовища. Готовийдо використання для великих програм.

Isabelle включає в себе потужні специфічні інструменти, , наприклад типи даних, індуктивні визначення та рекурсивні функції

Доведення проводяться в структурованій мові доказів Isar, що використовує текст зрозумілий для людей і комп'ютерів.

Coq [14].Coq — інтерактивний програмний засіб доведення теорем, використовує власну мову функціонального програмування (Gallina) з залежними типами. Дозволяє записувати математичні теореми і їх доведення, зручно модифікувати їх, перевіряє їх на правильність. Користувач інтерактивно створює доказ зверху вниз, починаючи з мети (тобто від гіпотези, яку необхідно довести). Coq може автоматично знаходити докази в деяких обмежених теоріях за допомогою так званих тактик. Coq застосовується для верифікації програм.

Coq розроблений у Франції в рамках проєкту TypiCal (раніше — LogiCal), спільно зкерованим INRIA, Політехнічною школою, Університетом Париж-південь XI та Національним центром наукових досліджень, раніше була виділена група і у Вищій нормальній школі Ліона.

Два перших інструменти багаторазово застосовувалися для верифікації як програмних, так і апаратних систем.

Однією з найбільш важливих проблем у символьному моделюванні є проблема побудови канонічних і нормальних форм логічних формул. Так, незважаючи на те, що кількість станів символьної моделі, які задаються

формулою менше, ніж конкретної, в символьному моделюванні виникають складні питання визначення досяжності вже пройдених станів у процесі використання динамічних методів і досить складні формули, складність яких через відсутність деякої канонічної або нормальної форми може рости крок за кроком.

На даний час, існує досить багато спроб реалізації інструментів, які використовуються для вирішення даної проблеми.

В роботі розглянуто такі системи побудови доказів як CVC4, MathSAT5, QEPCAD, Singular, COCOA, MiniZinc, STP, RedLog, Satallax, Isabelle, E-SETHEO, Minisat, SMTInterpol, TPS / ETPS, Paradox, Gandalf, Vampire.

Основну увагу при аналізі розглянутих ситем було зосереджено на наявності інструментів спрощення формул. Нижче подано короткі характеристики розглянутих систем і результати їх випробувань. Формули, які використовувалися для перевірки інструментів спрощення, представлено в Додатку 2.

CVC4 1.4. CVC4 є ефективним з відкритим вихідним кодом автоматичним доказувачем теорем для SMT проблем. Може бути використаний для доказу справедливості (або виконуваності) формул першого порядку завдяки великому числу вбудованих логічних теорій та їх комбінацій. Особливостями CVC4 є, то що він працює з версією логіки першого порядку з поліморфними типами і має широкий спектр функцій, включаючи декілька вбудованих базових теорій: раціональну та цілочисельну лінійну арифметику, масиви, кортежі, записи, індуктивні типи даних, бітові вектори, рядки і рівності над неінтерпретованими функціональними символами; підтримку кванторів; інтерактивний текстовий інтерфейс; багатий C ++ API для вбудовування в інші системи [15].

Для спрощення формул у прувері використовується метод TRANSFORM. Для отримання нормальних результатів виникає необхідність укладення формул в дужки, що говорить про проблему з реалізацією пріоритету операцій. Більше того, застосування методу спрощення робить формули більш громіздкими (квантор існування завжди замінюється на квантор спільності, з'являється багато символів заперечення, не може знімати квантори в незамкненій формулі).

Позитивною характеристикою прувера є можливість використання його в розроблюваних проєктах на C ++, а також наявність документації та більш-менш зручний і зрозумілий для користувача синтаксис.

MathSAT5 5.3.10 - ефективний SMT вирішувач, наступник MathSAT 4. Підтримує широкий спектр теорій (в тому числі, наприклад, рівності і неінтерпретовані функції, лінійну арифметику, бітові вектори та масиви) і функціональних можливостей (в тому числі, наприклад, обчислення інтерполянтів Крейга, генерацію та доказ моделі) [16].

У якості форматів введення підтримує SMT-LIB 2, SMT-LIBv1.2, DIMACS. Працює з GMP (Gnu Multiprecision library) і GNU C Library. Можна використовувати в C / C ++ проєктах. Є перевірка SAT. Формат введення обмежено бібліотеками, які інструмент використовує в якості формату введення. Легко встановлюється і має хорошу документацію. Інструменти спрощення формул відсутні, відповідно не може використовуватися для вирішення поставленої задачі.

QEPCAD Version B 1.69 – інтерактивна програма командного рядка, написана на C / C ++, а також на основі бібліотеки SACLIB. QEPCAD і бібліотека SACLIB є результатом програми досліджень Джорджа Коллінза і його аспірантів [17]. Дещо невдалою особливістю бібліотеки SACLIB, на якій засновано QEPCAD є те, що вона ініціалізується за фіксованим розміром шматка пам'яті і при перезаповненні видає повідомлення про помилки. Однією з найбільш важливих особливостей QEPCAD є його здатність генерувати прості безкванторні еквівалентні формули [17], що говорить проможливості його використання для спрощення формул.

Гарна система з точки зору спрощення формул. Крім стандартних кванторів: кванторів існування "E" і спільності "A", підтримує додаткові квантори: "F" - для нескінченно багатьох; "G" - для всіх але скінченно багатьох; "C" - для приєднаної множини; "xk" -для точної кількості k різних значень.

Її недоліками, на наш погляд, є те, що:

1) QEPCAD не розуміє пріоритету застосування логічних операторів;

2) не підтримує функціональні символи (потрібно використовувати алгоритми заміни функціоналів на змінні, що робить формулу ще більш громіздкою);

3) використовує різні дужки (() - для числових операцій [] - для логічних операцій);

4) від'ємні числа потрібно брати в дужки;

5) відсутній знак множення (5 * x => 5x), що не завжди зручно. Справляється з формулами: 1-9,15-17 (див. Додаток 2).

Singular 4.0.3 – система комп'ютерної алгебри для поліноміальних обчислень, з особливим акцентом на комутативну і некомутативну алгебри, алгебраїчну геометрію і теорію особливостей. Безкоштовна система з відкритим вихідним кодом під ліцензією GNU General Public Licence [18].

Система легко встановлюється, відмінно документована, має хороший синтаксис. Є обчислення інваріантів. Може бути використана як система комп'ютерної алгебри, але не підходить для вирішення поставленого завдання, тому що не розвинена як система для роботи з математичною логікою (відсутні квантори; не можна використовувати нерівності для логічних формул, відсутні перевірки на SAT, доказів).

COCOA 5.1.3. CoCoA (Computations in Commutative Algebra) [19] вільна система комп'ютерної алгебри для обчислень з числами і многочленами. Бібліотека CoCoA (CoCoALib [20]) доступна відповідно до GNU General Public License. Було перенесено на багато операційних ситем, включаючи Macintosh на PPC і x86, Linux на x86, x86-64 & PPC, Solaris на SPARC і Windows на x86. В основному використовується дослідниками, але може бути корисна і для "простих" обчислень. Особливостями CoCoA є: Дуже великі цілі та раціональні числа (завдяки використанню GNU Multi-Precision Library), різноманітні поліноми, базиси Гребнера. Як і Singular, CoCoA не має функціоналу для роботи з математичною логікою (не підтримує квантори, не спрощує формули). Відповідно, не може бути використана для вирішення поставлених завдань.

MiniZinc (WIndows) – середньорівнева мова констрейтного програмування. Є підмножиною Zinc - мови більш високого рівня. Модель MiniZinc не диктує спосіб вирішення проблеми, хоча модель може включати анотації, які використовуються для управління вирішувачем, що лежить в основі. MiniZinc розроблено для легкої взаємодії з різноманітними внутрішніми вирішувачами, що досягається шляхом перетворення вхідної моделі MiniZinc та файла даних в модель FlatZinc [21]. Написано на C ++. Працює лише з цілими та раціональними числами. Цікавий інструмент для моделювання і констрейтного програмування. Легко встановлюється і має хорошу документацію. Має також IDE, що полегшує роботу з ним. Після задання обмежень на змінні у моделі (констрейтне програмування), знаходить рішення, яке задовольняє дані обмеження, за допомогою таких функцій як: solve satisfy (будь-яке рішення), solve maximize (максимізувати функцію), solve minimize (мінімізувати функцію). Окрім цього, не виконує необхідні спрощення. Відповідно, на нашу думку, є гарним інструментом в області моделювання та констрейтного програмування, але не підходить для поставленої задачі спрощення формул.

STP 2.1.2. STP - вирішувач з обмеженнями, який може вирішувати багато видів проблем, з використанням інструментів аналізу програм, доказувачів теорем, автоматичних засобів пошуку помилок, криптографічних алгоритмів та засобів перевірки моделей. Ефективно розвивається та використовується багатьма компаніями та дослідницькими інститутами. Є ефективною процедурою прийняття рішень щодо валідності/виконуваності формул з безквантворної багатосортної теорії бітових векторів фіксованої ширини та (неекзистиціональних) одномірних масивів. Функції мови введення STP включають конкатенацію, виключення, зсув вліво/вправо, унарний мінус, додавання, множення, (підписаний) за модулем / діленням, побітові логічні операції, умови if-then-else, читання та запис в масив [22].

Підтримує CVC, SMT-LIB1, і SMT-LIB2 формати введення даних. Рекомендовано використовувати SMT-LIB2 (хоча не всі її можливості використовуються в STP). Працює з бітовими векторами та масивами. Не вміє

працювати з кванторами. Є SAT-перевірка. Має спрощувач, який містить алгоритми спрощення, але не працює у необхідному режимі - задали формулу → отримали спрощену. Написано на C, отже є можливість інтеграції в C ++ код.

RedLog (Reduce _2015_10_20 Windows). є невід'ємною частиною інтерактивної комп'ютерної алгебраїчної системи Reduce. Доповнює всебічну колекцію потужних методів із символьних обчислень, надаючи понад 100 функцій для формул першого порядку. Redlog публічно доступний з 1995 року і постійно вдосконалюється [23].

Redlog, як правило, працює з інтерпретованою логікою першого порядку на відміну від логіки першого порядку. Кожна формула першого порядку в Redlog повинна містити виключно атоми одного конкретного домену, що підтримується Redlog, що відповідає вибору допустимих функцій та зв'язків із фіксованою семантикою. Домени, що підтримуються Redlog, включають нелінійну реальну арифметику (алгебру Тарського), арифметику Пресбургера, параметричні кількісно визначені булеві формули та багато іншого. І Reduce, і Redlog є відкритими та вільно розповсюджуються під дуже ліцензією FreeBSD [23].

Є квантори спільності та існування. Система має досить зручний та зрозумілий синтаксис, докладну документацію, легко встановлюється. Присутні функції, квантори та пріоритет операцій. Спрощує лише формули 15,16 (див. Додаток 1). Інші формули просто переписуються в іншому вигляді. Відсутні факторизация, SAT перевірка та доказ формул. Система написана на LISP, відповідно інтеграція з C ++ системами ускладнена.

Satallax 2.8. Satallax - це автоматизований доказувач теорем для логіки вищого порядку. Особливою формою логіки вищого порядку, яку підтримує Satallax, є проста теорія типу Черча з операторами розширення та вибору. Вирішувач SAT MiniSat відповідає за більшу частину пошуку доказів. Satallax генерує пропозиційні вирази відповідно до правил таблиці обчислень, і періодично викликає MiniSat, щоб перевірити виконання цих пунктів. Satallax реалізований в Objective Caml [24].

Satallax використовує вирішувач SAT MiniSat як інструмент для перевірки поточного набору пропозиційних виразів на незадовільність. Якщо вирази є незадовільними, то вихідний набір формул вищого порядку є незадовільним. Якщо у типів функцій немає кванторів, генерація формул вищого порядку та відповідних виразів може припинитися. У такому випадку, якщо MiniSat повідомляє остаточний набір виразів як задовільний, то вихідний набір формул вищого порядку є задовільним [24].

Підтримує логіку вищого порядку. Як формат вводу даних використовує THF мову TPTP бібліотеки. Систему написано на Ocaml, тому її інтеграція в розроблювані C++ системи ускладнена. Відсутня документація. Є лише короткий опис інструменту. Для SAT перевірки використовує систему MiniSat. MiniSat є відкритим ресурсом і його можна використовувати в C ++ коді, так як система написана на C ++. Програма виконує свої завдання SMT-прувер, але не орієнтована на спрощення предикатних формул.

Isabelle2016 (Windows). Стислий опис доказувача надано вище. Стосовно результатів експерименту, можна сказати наступне:

Легко встановлюється, має докладну документацію. Підтримує роботу з логіками першого та вищого порядку.

Є гарною системою з точки зору спрощення формул. Користувач сам задає процес правила для спрощення формул, а вже за правилами користувача спрощуються формули (працює як система переписувальних правил). Відповідно, відповідальність за результат спрощень лежить на самому користувачеві. За замовчуванням, як аналогічні інструменти - QEPCAD або CVC4, формули не спрощує. Написано на мові SML, тому інтеграція даної системи дуже важка з C ++.

E-SETHEO. E-SETHEO поєднує в собі найрізноманітніші доказувачі теорем та спеціалізовані процедури прийняття рішень в одну з найпотужніших ATP систем. Основна ідея фреймворку E-SETHEO полягає в тому, щоб дозволити різним процедурам пошуку доказів конкурувати за ресурси для вирішення даної проблеми. E-SETHEO використовує безліч різних механізмів виведення,

побудованих на різних обчисленнях та використовуючи різні евристики. Існує два різних режими роботи: режим навчання та режим застосування. Крім того, E-SETHEO також використовує спеціальні процедури прийняття рішень щодо пропозиційних та скінчено обґрунтованих проблем, а також стратегії кооперації, побудовані на обміні лемами між різними системами [25].

На жаль, інструмент не працює. Головною проблемою використання інструменту є те, що проєкт було реалізовано відразу на багатьох мовах програмування C, Eclipse Prolog, Perl, Bigloo Scheme, та не доопрацьовано (на даний час не підтримується). В якості базису використовується E prover, написаний на ANSI-C, який можна використовувати для власних проєктів.

Minisat 2.2.0. MiniSat - це мінімалістичний вирішувач SAT з відкритим кодом, розроблений для того, щоб допомогти дослідникам та розробникам розпочати роботу з SAT. MiniSat розпочав свою діяльність у 2003 році, намагаючись допомогти розробниками приєднатися до спільноти SAT, надавши невеликий, але ефективний вирішувач SAT з гарною документацією [26].

Реалізовано на C ++, має MIT ліцензію. Можна вільно брати вихідні файли та працювати з ними. Приймає введення даних в форматі DIMACS (CNF - формат вводу даних). Вихідні тексти нормально компілюються безпомилково, гарний консольний прувер. Спрощення формул не виконує, тому формат введення даних повинен бути відразу в кон'юктивній нормальній формі. Є звичайним SAT-вирішувачем з незначним функціоналом.

SMTInterpol 2.1. SMTInterpol - це вирішувач SMT, який може обчислювати інтерполянти Крейга для різних теорій. Розроблено на кафедрі програмної інженерії Фрайбурзького університету. Розв'язувач написано на Java і може використовуватися в будь-якій операційній системі, що підтримує Java (починаючи з 6-ї версії). Поточна версія SMTInterpol підтримує інтерполяцію для комбінації бесквантворних фрагментів теорій неінтерпретірованних функцій, лінійної дійсної арифметики, лінійної цілочисельної арифметики та масивів [27].

Використовує SMT-LIB як формат вводу даних. У разі потреби, можна адаптувати для C ++. Є проста документація на сайті і покликання на наукові публікації. Легко встановлюється та запускається. Підтримує функції.

Перевіряє SAT. Має експериментальний спрощувач формул. Експериментальний спрощувач не спрацював з жодною з формул. Не працює з кванторами. На наш погляд, має мало можливостей та є слабким прувером.

TPS/ETPS. TPS та ETPS – це, відповідно, дві системи - система доведення теорем та навчальна система доведення теорем. Перша - це автоматизований доказувач теорем для логіки першого порядку та теорії типів. Друга - це скорочена версія TPS, призначена для використання студентами (містить лише команди, що стосуються інтерактивного доведення теорем). TPS та ETPS працюють у Common Lisp і можуть бути використані в будь-якій системі, де працює Common Lisp. TPS та ETPS широко використовуються в системах Unix та Linux, а певною мірою і в Windows [28].

Потенційні програми автоматизованого доведення теорем включають перевірку апаратного та програмного забезпечення, часткову автоматизацію різних математичних видів діяльності, сприяння розвитку формальних теорій у найрізноманітніших дисциплінах, дедуктивні інформаційні системи для цих дисциплін, експертні системи, які можуть думати, та деякі аспекти штучного інтелекту .

На жаль, системи з офіційного сайту не завантажуються, а вихідні файли з CASC не компілюються, що робить неможливим використання інструменту та проведення експерименту щодо спрощення формул. Системи написано на Common Lisp, тому їх інтеграція в C ++ системи ускладнюється.

Paradox 3.0. Paradox - це автоматизована система доведення теорем, розроблена Коеном Ліндстремом Классеном та Нікласом Серенсонсом з Технологічного університету Чалмерса. Програмне забезпечення написане мовою програмування Haskell, є безкоштовним і випускається на умовах Загальної публічної ліцензії GNU [29].

Знаходиться лише в загальнодоступньому каталозі пруверів попередніх CASC випробувань. Покликання на офіційний сайт не працює. Відсутня інформація щодо пруверу і в Hackage (репозиторії програм і бібліотек для Haskell), що свідчить про те, що розробку даного прувера припинено. Файли з CASC не компілюються.

Gandalf c-2.6.r1. це автоматизований доказувач теорем першого порядку, що застосовується до кількох конкретних доменних завдань, таких як Семантична павутина. Брав участь у змаганнях CADE ATP System Competition. Розроблено на мові програмування Scheme, яка потім компілюється до мови програмування C, використовуючи Hobbit від SCM [30]. Доводить теореми, сформульовані в логіці. Оскільки логіка є досить універсальною мовою, системи ATP, такі як Gandalf, можуть довести теореми з математики та перевірити складні системи, такі як цифрові схеми, програмне забезпечення та протоколи зв'язку. Новою областю застосування для систем ATP є Semantic Web: проєкт, спрямований на розміщення машинозрозумілого вмісту в Інтернеті.

Гендальф розробляється з 1995 року в Талліннському технічному університеті. Раніше брав участь у щорічних змаганнях CASC для ATP систем. Доступ до вихідних файлів і документації закрито для тих, у кого немає доступу до аккаунту Талліннського університету. Останнє датування прувера відзначено 2003 роком. У змаганнях останніх років участі не приймали, відповідно, можна зробити висновок, що розробку інструменту зупинено. Так як прувер написано на Scheme, він не підходить для інтеграції в розроблювані на с ++ системи.

Vampire 2.6 (Vampire 4.0). VAMPIRE це автоматичний доказувач теорем для логіки першого порядку. Використовувався в ряді академічних та промислових проєктів. Перша версія VAMPIRE була впроваджена в 1993 році, потім його кілька разів було переписано. Реалізацю поточної версії розпочато у 2009 році. Реалізовано на C ++ [31].

VAMPIRE реалізує обчислення впорядкованої двійкової роздільної здатності та суперпозиції для обробки рівності. Він також реалізує обчислення Inst-gen та конструктор скінченних моделей у стилі MACE. Внутрішньо

VAMPIRE працює лише з казуальною нормальною формою. Проблеми усуваються під час попередньої обробки [32].

На офіційному сайті сказано, що прувер можна буде завантажити після виходу 3-ї версії (хоча в CASC бере участь версія 4.0). Попередні версії скачати неможливо. Вихідні файли Vampire з загального каталогу пруверів CASC змагань не компілюються через помилки, пов'язані з вихідним кодом. Використовувати можливості інструменту у власному софті забороняється ліцензією по отриманню прувера з офіційного сайту. Сайт має видавати лише скомпільований варіант прувера у вигляді бінарних файлів. Відповідно, хоча за описом прувер має багато можливостей, випробувати його неможливо.

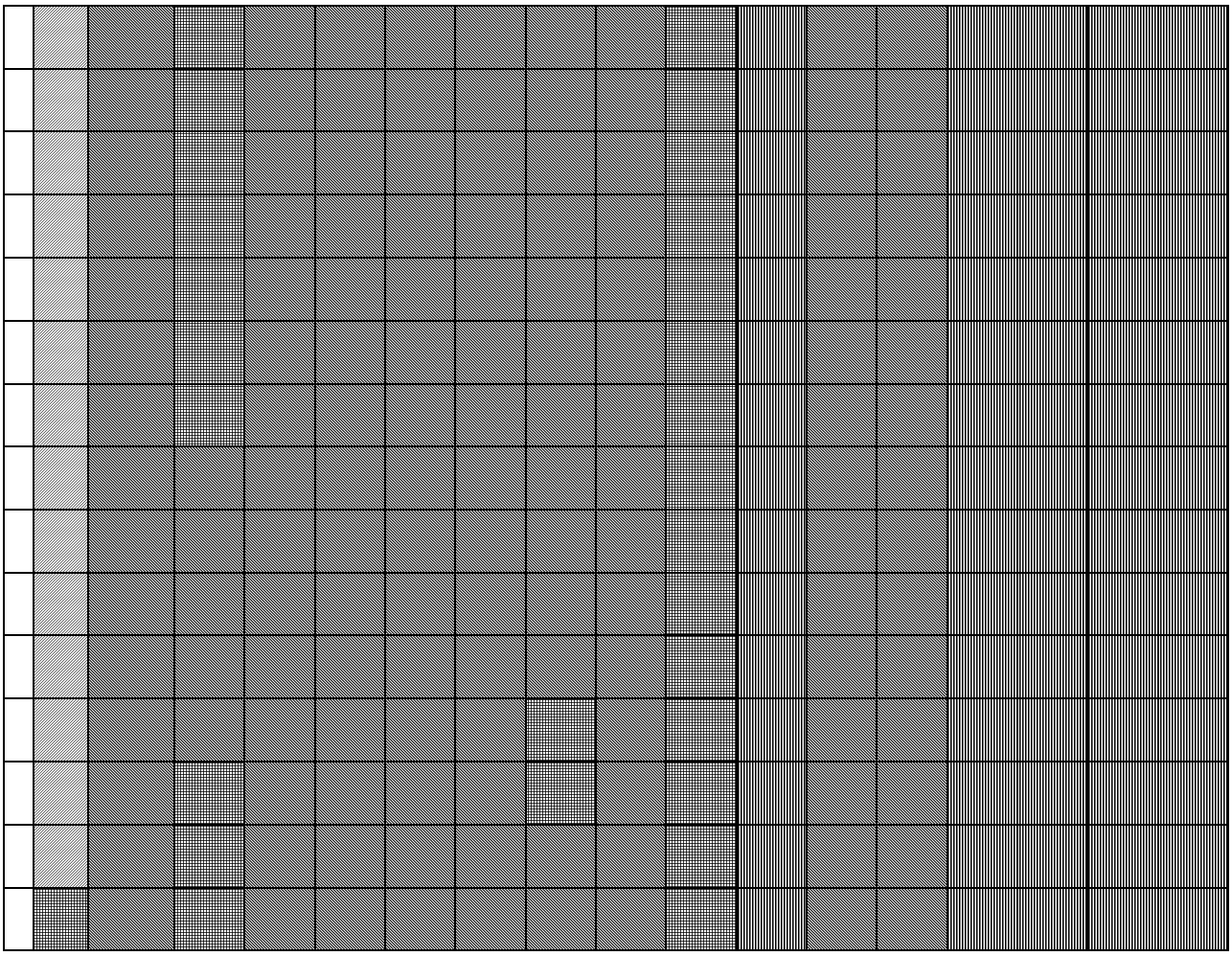
Як можна побачити з матеріалу описаного вище практично всі розглянуті системи запустилися і працювали. Виходячи з необхідності витрати великої кількості часу на їх налаштування, встановлення та вивчення в огляд не було включено й інші, не менш важливі на наш погляд системи: E.TheoremProver (http://wwwlehre.dhbw-stuttgart.de/~sschulz/E/E.html), iProver (http://www.cs.man.ac.uk/~korovink/iprover/), LEO-II (http://page.mi.fu-berlin.de/cbenzmueller/leo /), Z3 (https://github.com/Z3Prover/z3), WaldMeister (http://www.waldmeister.org/), SPASS. (Http://www.mpi-inf.mpg.de/departments/automation-of-logic/software/spass-workbench/).

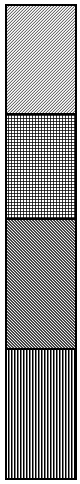
Формули, які використовувалися для перевірки розглянутих ресурсів можна переглянути в Додатку 2. Результати спрощень представлено в Таблиці 3.1.

Таблиця 3.1

Результати спрощення формул SMT вирішувачами

	CVC4 1.4	**MathSAT 5.3.10**	**1.69**	**2016**	**COCOA 5.1**	**MiniZinc 2.0.12**	**STP 2.1.2**	**(Reduce**	**Satallax 2.8**	**Isabelle2016**	**E-SETHEO**	**Minisat 2.2.0**	**2.1**	**TPS/ETPS 3.0**	**Paradox 3.0**	**Gandalf c-2.6.r1**	**(4.0)**





Формат спрощення не є задовільним

Відмінно пройшов випробування, формула була спрощена

Не зміг спростити формулу або не має програмних можливостей для її спрощення

Результат не відомий, так як з тих чи інших технічних або інформаційних причин нам не вдалося скомпілювати, запустити прувер або знайти до нього документацію, яка декларує його можливості і пояснює його використання

Як видно з Таблиці 3.1, повністю з поставленим завданням впоралася лише одна система - Isabelle2016 [13]. Система легко встановлюється і має хорошу документацію. Підтримує роботу з логікою першого порядку і вищого порядку.

Хороша сиситема з точки зору спрощення формул. Правила спрощення формул задаються користувачем. В цьому плані працює як система переписувальних правил. Тому відповідальність спрощень лежить на самому користувача. Написана на мові SML, тому інтеграція даної системи дуже важка з C ++.

Що стосується QEPCAD [17], - з точки зору спрощення формул - система працює добре, але має досить складний і громіздкий синтаксис і не містить хорошої документації. Найбільш використовуваними мовами для створення подібних систем є C / C ++ і ML-мови (OCaml, SML). У всіх системах ми стикаємося з проблемою необхідності переписування формул в певному форматі. Переписувати формули і приводити їх до необхідного формату дуже важко, особливо, якщо всі операції потрібно виділяти дужками, як, наприклад, в SMT-LIB. Що стосується таких інструментів як E-SETHEO [3], TPS / ETPS [28], Paradox 3.0. [29], Gandalf c-2.6.r1 [30], Vampire 2.6 [31], можна зробити висновок, що частина з них є закритими проєктами (E-SETHEO, Gandalf c-2.6.r1), до інших пруверів закрито доступ. Вихідний код перерахованих пруверів з CASC містить помилки, виходячи з чого, інструменти випробувати не вдалося.

Відповідно, розробка системи спрощення формул, яка дозволяла б вирішити поставлені завдання залишається актуальною. Крім того, не менш важливою є і необхідність реалізації відповідних алгоритмів, що дозволяють виконувати спрощення формул на більш ефективному рівні.

3.2. Канонічні форми лінійних напівалгебраїчних формул.

Введемо основні позначення:

$X = \{x_1,...,x_n\}$ - n - вимірний вектор змінних.

Q - конструктивне лінійно впорядковане поле, назване полем коефіцієнтів. Q^n - векторний простір над Q.

$\bar{a} = (a_1,...,a_n) \in Q^n$ - вектор числових коефіцієнтів, $b \in Q$ - числовий коефіцієнт.

$LF(\bar{a}, X)$ -лінійна форма від $X, \bar{a}$: $LF(a, X) \overset{df}{=} a_1x_1 + ... + a_nx_n$

Лінійна нерівність (ЛН) $E(\bar{a}, X, c)$ має вигляд $LF(\bar{a}, X) \leq c$ і позначається через $E(\bar{a}, X, c)$. Для скорочення запису лінійні нерівності іноді будемо записуватися у вигляді $E(X)$, вказуючи тільки змінні, від яких воно залежить.

На даний момент існує велика кількість підходів до вирішення лінійних нерівностей, на основі яких реалізовано відповідні алгоритми в існуючих пруверах та солверах.

В залежності від підходів та цілей умовно їх можна розділити на такі групи:

- Алгоритми, похідні від алгоритмів Фурьє Моцкіна [33].
- Симплекс-метод та його похідні.
- Циліндрична алгебраїчна декомпозиція [34].
- Швидкі алгоритми для формул певного виду.

Алгоритми Фурьє Моцкіна є першою вирішуючою процедурою для арифметики дійсних чисел. Пізніше цей метод було адаптовано для розвязання арифметики Пресбургера, наприклад алгоритм Omega Test [35]. До недоліків алгоритму відноситься суперекспоненціальна складність програм у найгіршому випадку.

Симплекс метод підходить для перевірки наявності розвязку системи лінійних нерівностей, рівностей та заперечень рівності, не включаючих квантори. Демонструє поліноміальний середній час обчислень. Симплекс метод застосовується для швидкої перевірки сумісності системи. Інкрементальний симплекс метод [36] дозволяє ефективно перевіряти сумісність нового обмеження, доданого в консистентну систему обмежень. Цю властивість можна використовувати на етапі мінімізації проміжних формул. Оскільки даний метод не може спрощувати формулу шляхом елімінації кванторів, не може використовуватися як основний алгоритм для поставлених задач.

Алгоритм циліндричної алгебраїчної декомпозиції, розроблений Колінсом [34], дозволяє вирішувати проблеми елімінації кванторів будь-якої складності, але зважаючи на високу обчислювальну складність його практичне застосування значно обмежено.

Алгоритми для формул певного виду – дозволяють розвязувати задачу парктично за лінійний час. Такими формулами, наприклад, можуть бути формули які складаються лише з двох змінних в кожній системі обмежень, або, іншими словами – коньюнкції нерівностей [36].

Визначення 3.1. Лінійною напівалгебраїчною формулою (ЛПФ) над Q від X називається безквантерна логічна формула $F(X) = F(E_1(\bar{a}_1, X, c_1),...,E_m(\bar{a}_m, X, c_m))$ - від лінійних нерівностей $E_1(X),...E_m(X)$ в сигнатурі логічних зв'язок $\&,\vee$. Лінійною напівалгебраїчною множиною $M(F)$ називається множина $M(F)) = \{V \in Q^n \mid F(V)\}$.

Іншими словами, лінійна напівалгебраїчна формула - це сукупності і системи лінійних нерівностей, а лінійна напівалгебраїчна множина - багатогранна область в Q^n - інтерпретація цієї формули.

Визначення 3.2. Нехай x - змінна, що входить в ліву частину ЛН з ненульовим коефіцієнтом і $Y = X - \{x\}$.

Тоді ЛН можна перетворити до вигляду

$$x \le LF(\bar{b}, Y) + c \text{ або } x \ge LF(\bar{b}, Y) + c \qquad (3.1).$$

Таку форму ЛН будемо називати дозволеною щодо x або x- дозволеною. Якщо кожне ЛН лінійної напівалгебраїчної формули, залежне від x, представлено в дозволеній формі, то ЛПФ будемо називати представленою в x- дозволеній формі.

Перетворивши формулу $F(E_1,...,E_m)$ за правилами алгебри висловлювань обчислення диз'юнктивно-нормальної форми, можна отримати її представлення у вигляді сукупності систем ЛН. Однак, таке подання не є єдиним і тому погано пристосоване для алгоритмів комп'ютерної алгебри.

Визначення 3.3. Формулою x-сегмента $s = [L(Y), x, R(Y)]$ над множиною змінних $Y, x \notin Y$, будемо називати подвійну лінійну нерівність виду $L(Y) \le x \le R(Y)$.

Якщо $Y = \{y_1,....,y_m\}$, формула $s = [L(Y), x, R(Y)]$ інтерпретується на просторі Q^{m+1} як елементарна двогранна область $M(s) = \{(q_0, \bar{q}) \in Q^{m+1} \mid L(\bar{q}) \le q_0 \le R(\bar{q})]\}$ - x-сегмент.

Нерівності, представлені в x-дозволеній формі виду $x \le R(Y)$ або $x \ge L(Y)$, можна перетворити в x- сегменти: $s = [-\infty, x, R(Y)]$ або $s = [L(Y), x, +\infty]$.

Нехай $M \subset Q^m$ и $s = [L(Y), x, R(Y)]$. Визначимо обмеження $s \cdot M$ сегмента s на M наступним чином: $s \cdot M = [L(Y), x, R(Y)] \cdot M \overset{df}{\approx} (L(Y) \le x \le R(Y)) \,\&\, (Y \in M)$. $s \cdot M \ne \varnothing$ тоді і тільки тоді, коли виконується умова $\forall \bar{q}(\bar{q} \in M) \to (L(\bar{q}) \le R(\bar{q}))$. Цю умову будемо записувати у формі $L(x) \le_M R(x)$.

Очевидно, що має місце властивість

$$M \cdot [L, x, R] \cdot M' = M \cdot [L, x, H] \cdot M' + + M \cdot [H, x, R] \cdot M' \qquad (3.2)$$

Співвідношення (3.2) визначає на множині сегментів операцію приведення і зворотну їй операцію розбиття множини на частини. Точне визначення розбиття наведено нижче.

Системи лінійних нерівностей і трапецоїди

На змінних вектора $X = \{x_1, ..., x_n\}$ визначимо порядок $x_1 < ... < x_n$ і позначимо $X_j \overset{df}{=} \{x_j, ..., x_n\}$.

Нехай $s_j = [L_j(X_{j-1}), x_j, R_j(X_{j-1})], j = 1, ..., n-1$, $s_n = [L_n, x_n, R_n], L_n, R_n \in Q$.

Визначення 3.4. Множина T, визначена формулою $T = s_1 \cdot s_2 \cdot ... \cdot s_n$ (дужки групуються за замовчуванням вправо), називається трапецоїдом в просторі Q^n.

Через T_j позначимо трапецоїд $s_j \cdot s_{j+1} \cdot ... \cdot s_n$. Тоді $T = T_1$, $T_j = s_j \cdot T_{j+1}, j = 1,2,..., n-1$, причому T_{j+1} - проекція T_j на підпростір Q^{n-j+1}, визначений координатами $\{x_{j+1}, ..., x_n\}$.

Таким чином, трапецоїд визначено послідовністю проекцій на спадаючу послідовність підпросторів $Q^n \supset Q^{n-1} \supset ... \supset Q^1$, причому кожна проекція - також трапецоїд.

Подання трапецоїда T у вигляді послідовності його проекцій будемо позначати формулою

$$T = T^{(n)} \to T^{(n-1)} \to ... \to T^{(1)}.$$

Випукла множина $P \subseteq Q^n$, яка представляє рішення системи лінійних нерівностей, називається полігоном.

Визначення 3.5. Розбиттям множини $A \subset Q^n$ будемо називати її представлення у вигляді такого об'єднання підмножин $A = A_1 + + A_2 + + ... + + A_k$, що $i \neq j \to Dim(A_i \cap A_j) < n$.

Це означає, що різні елементи розбиття перетинаються по множині, розмірність якої менше розмірності простору.

Полігон P може бути представлено у вигляді розбиття $P = T_1 + + ... + + T_m$ на трапецоїди $T, ..., T_m$.

Алгоритм описано в [37].

Визначення 3.6. Розбиття $P = T_1 + + ... + + T_m$ будемо називати непривідним, якщо до будь-якої пари трапецоїдів T_i, T_j можна застосувати операцію приведення (3.2).

Впорядкування елементів розбиття полігону на трапецоїди. Непривідне розбиття полігону на трапецоїди єдине з точністю до перестановок трапецоїдів розбиття. Впорядкування послідовності можна здійснити відповідно до таких його властивостей. Якщо $s_1^{(n)}, ..., s_m^{(n)}$ - послідовність проекцій непривідного розбиття полігону $P = T_1 + + ... + + T_m$ на вісь Ox_n, то або $s_i^{(n)} = s_j^{(n)}$, або $Dim(s_i^{(n)} \cap s_j^{(n)}) = 0$, тобто $s_i^{(n)}, s_j^{(n)}$ або збігаються, або не перетинаються, або є суміжними. Тому $s_1^{(n)}, ..., s_m^{(n)}$ можуть бути переставлені так, щоб бути розміщеними на осі Ox_n в порядку зростання. Якщо $s_{i_1}^{(n)}, ..., s_{i_l}^{(n)}$ - попарно різні числові сегменти, то впорядкування має вигляд

$$s_1^{(n)} = ... = s_{i_1}^{(n)} = [L_1, x_n, L_2],\ s_{i_1+1}^{(n)} = ... = s_{i_2}^{(n)} = [L_2, x_n, L_3], ...,\ s_{i_{l-1}+1}^{(n)} = ... = s_m^{(n)} = [L_l, x_n, L_{l+1}],$$

де $L_1 < L_2 < ... < L_{l+1}$. Тоді полігон P можна представити у вигляді

$$P = (T_1 + + ... + + T_{i_1}) + + (T_{i_1+1} + + ... + + T_{i_2}) + + ... + + (T_{i_l+1} + + ... + + T_m)$$

або

$$P = (T_1^{(n-1)} + + ... + + T_{i_1}^{(n-1)}) \cdot s_{i_1}^{(n)} + + (T_{i_1+1}^{(n-1)} + + ... + + T_{i_2}^{(n-1)}) \cdot s_{i_2}^{(n)} + + ... + + (T_{i_l+1}^{(n-1)} + + ... + + T_m^{(n-1)}) \cdot s_m^{(n)}$$

Позначимо розбиття трапецоїдів в дужках через $P_{i_1}^{(n-1)}, P_{i_2}^{(n-1)}, ..., P_m^{(n-1)}$.

Тоді $P = P_{i_1}^{(n-1)} \cdot s_{i_1}^{(n)} + + P_{i_2}^{(n-1)} \cdot s_{i_2}^{(n)} + + ... + + P_m^{(n-1)} \cdot s_m^{(n)}$.

Властивість розбиття полігону, описана вище, має місце для проекцій будь-якої розмірності. Саме, нехай $P = T_1 + + ... + T_l$ - непривідне розбиття полігону із загальною $k+1$-ю проекцією $T^{(k+1)}$:

$$T_1 = T_1^{(1)} \to ... \to T_1^{(k)} \to T^{(k+1)}, ..., T_l = T_l^{(1)} \to ... \to T_l^{(k)} \to T^{(k+1)}.$$

Тоді $T, ..., T_l$ можна впорядкувати таким чином, щоб виконувалася властивість *канонічності*:

$$s_1^{(k)} = ... = s_{i_1}^{(k)} = [L_1, x_k, L_2],\ s_{i_1+1}^{(k)} = ... = s_{i_2}^{(k)} = [L_2, x_k, L_3], ...,\ s_{i_{l-1}+1}^{(k)} = ... = s_m^{(k)} = [L_l, x_k, L_{l+1}], \quad (3.2.1)$$

$$L_1 <_{T^{(k+1}} L_2 <_{T^{(k+1}} ... <_{T^{(k+1}} L_{l+1}. \quad (3.2.2)$$

Непривідне розбиття полігону на трапецоїди, впорядковане відповідно до властивості канонічності, будемо називати канонічним розбиттям або канонічною сумою.

Оскільки $T_i^{(j)}$ - проекція $T_i^{(j+1)}$ на підпростір Q^j, визначення 3.5 означає, що в канонічній сумі будь-які проекції трапецоїдів-складових можуть перетинатися тільки по їх межі. Співвідношення (3.2) можна назвати операцією приведення подібних.

Теорема 3.1. Нехай $L_1 \& L_2 \& ... \& L_m$ - система лінійних нерівностей і $P \subset Q^n$ - полігон (опукла множина в Q^n), визначена цією системою. Тоді P можна представити у вигляді канонічної суми трапецоїдів $P = T_1 + + ... + + T_k$.

Алгоритм 1 (перетину трапецоїдів). На множині трапецоїдів у просторі Q^n природно визначити операцію перетину $T_1 \cap T_2$. Цей перетин можна представити у вигляді канонічної суми трапецоїдів. Розглянемо алгоритм обчислення перетину $T_1 \cap T_2 = T_1^{'} + + ... + + T_k^{'}$. Представимо T_1, T_2 у вигляді $T_1 = T_1^{(n-1)} \cdot s_{1n}, T_2 = T_2^{(n-1)} \cdot s_{2n}$. Тоді $T_1 \cap T_2$ обчислюється рекурсивно «знизу-вгору».

Базис рекурсії:

$$s_{1n} \cap s_{2n} = \varnothing \to T_1 \cap T_2 = \varnothing \quad else \quad T_1 \cap T_2 = (T_1^{(n-1)} \cap T_2^{(n-1)}) \cdot (s_{1n} \cap s_{2n}),$$

$$\max(L_{1n}, L_{2n}) \le_T \min(R_{1n}, R_{2n})] \to s_{1n} \cap s_{2n} = [\max(L_{1n}, L_{2n}), x_1, \min(R_{1n}, R_{2n})]$$
$$else\, s_{1n} \cap s_{2n} = \varnothing \qquad (3.3)$$

Крок рекурсії:

Припустимо, що $T_1 = T_1^{(n-k)} \cdot T_1^{(k)}, T_2 = T_2^{(n-k)} \cdot T_2^{(k)}$ та $T_1^{(k)} \cap T_2^{(k)}$ вже обчислено, причому $T_1^{(k)} \cap T_2^{(k)} = T_1^{'} + +... + + T_l^{'}$.

Тоді, з огляду на дистрибутивність позначки «++» щодо позначки «.», обчислення зводиться до обчислення $(T_1^{(n-k)} \cap T_2^{(n-k)}) \cdot T_j^{'}$, а це обчислення, в свою чергу, зводиться до обчислень $(s_{1n-k} \cap s_{2n-k}) \cdot T_j^{'}, j = 1,...,l.$ Результат операції повинен задовольняти співвідношенню умови в (3.3).

Оскільки $\max(L_1, L_2), \min(R_1, R_2)$ не є лінійними комбінаціями, результат операції перетину сегментів $s_{1n-k} \cap s_{2n-k}$ на $T_j^{'}$ залежить від взаємного розташування цих сегментів на трапецоїді $T_j^{'}$.

Співвідношення $L(Y) \le_T R(Y)$ визначимо наступним чином $L(Y) \le_T R(Y)$, якщо для будь-якого вектора $Y_0 \in T$ має місце нерівність $L(Y_0) \le R_k(Y_0)$. У канонічній формі представленя трапецоїда T ці співвідношення повинні виконуватися для будь-якого значення $k = n, n-1,...,1$ на підпросторах Q^k.

Розглянемо спочатку задачу обчислення перетину $s_1 \cap s_2$ на T в загальному вигляді. Нехай $s_1 = [L_1, x, R_1], s_2 = [L_2, x, R_2]$, $T_1 = s_1 \cdot T, T_2 = s_2 \cdot T$, причому $L_1 \le_T R_1, L_2 \le_T R_2$. Позначимо

$$LL_{ij} = T \cap (L_i \le_T L_j),\ RR_{ij} = T \cap (R_i \le_T R_j),\ LR_{ij} \overset{df}{=} L_i \le_T R_j,\ [LR]_{ij} \overset{df}{=} [L_i, x, R_j], i, j = 1,2 \qquad (3.4)$$

і перерахуємо співвідношення у вигляді переписуючих правил для всіх варіантів взаємного розташування нижніх і верхніх шапок T_1, T_2, записуючи формули (3.3) в термінах (3.4):

$$RR_{12} \to (\text{// } \textbf{варіанти 1-6}$$

1. $LR_{12} \to T_1 \cap T_2 = \varnothing$,
2. $(L_2 \cap R_1 \ne_T \varnothing)\ \&\ LL_{12} \to T_1 \cap T_2 = [LR]_{21} \cdot LR_{21}$,
3. $(L_2 \cap R_1 \ne_T \varnothing)\ \&\ (L_2 \cap L_1 \ne_T \varnothing) \to T_1 \cap T_2 = [LR]_{21} \cdot LL_{21}\ \&\ LR_{21} + +[LR]_{11} \cdot LL_{21}$,

4. $LR_{21}\,\&\,LL_{12}\to T_1\cap T_2=[LR]_{21}\cdot T$,

5. $LL_{21}\,\&\,(L_1\cap L_2\neq_T\varnothing)\to T_1\cap T_2=[LR]_{21}\cdot LL_{12}++[LR]_{11}\cdot LL_{21}$,

6. $LL_{12}\to T_1\cap T_2=T_2$);

$(R_2\cap R_1\neq_T\varnothing)\,\&\,LR_{12}\to$ (// **варіанти 7-11**

1. $(L_2\cap R_1\neq_T\varnothing)\,\&\,LL_{12}\to T_1\cap T_2=[LR]_{21}\cdot LR_{21}\,\&\,RR_{12}++[LR]_{22}\cdot RR_{21}$,

2. $(L_2\cap R_1\neq_T\varnothing)\,\&\,(L_2\cap_T L_1\neq\varnothing)\to T_1\cap T_2=[LR]_{21}\cdot LL_{12}\,\&\,LR_{21}++[LR]_{11}\cdot RR_{12}\,\&\,LL_{21}++[LR]_{12}\cdot RR_{21}$,

3. $LR_{21}\,\&\,LL_{12}\to T_1\cap T_2=[LR]_{21}\cdot RR_{21}++[LR]_{22}\cdot RR_{12}$,

4. $LR_{21}\,\&\,(L_1\cap_T L_2)\neq\varnothing)\to T_1\cap T_2=[LR]_{21}\cdot LL_{12}++[LR]_{11}\cdot LL_{21}\,\&\,RR_{12}++[LR]_{12}\cdot RR_{21}\,\&\,LL_{21}$,

5. $LL_{21}\to T_1\cap T_2=[LR]_{11}\cdot RR_{12}++[LR]_{12}\cdot RR_{21}$);

$(R_2\cap R_1\neq_T\varnothing)\,\&\,(R_2\cap L_1\neq_T\varnothing)\to$ (// **варіанти 12-14**

1. $(R_1\cap L_2\neq_T\varnothing)\,\&\,(L_1\cap L_2\neq_T\varnothing)\to T_1\cap T_2=[LR]_{21}\cdot RR_{12}\,\&\,LL_{12},++[LR]_{22}\cdot RR_{12}\,\&\,LL_{21}$
$++[LR]_{12}\cdot RR_{21}\,\&\,LL_{21}$,

2. $LR_{21}\,\&\,(L_1\cap L_2\neq_T\varnothing)\to T_1\cap T_2=[LR]_{21}\cdot RR_{12}\,\&\,LL_{12}++[LR]_{11}\cdot RR_{12}\,\&\,LL_{21}$
$++[LR]_{12}\cdot RR_{21}\,\&\,LL_{21}$,

3. $LL_{21}\to T_1\cap T_2=[LR]_{12}\cdot RR_{21}$);

$RR_{21}\,\&\,LR_{12}\to$ (// **варіант 15-17**

1. $LL_{12}\to T_1\cap T_2=T_2$,

2. $L_1\cap L_2\neq_T\varnothing\to T_1\cap T_2=[LR]_{22}\cdot LL_{12}++[LR]_{12}\cdot LL_{21}$,

3. $LL_{21}\to T_1\cap T_2=[LR]_{12}\cdot T$);

$RR_{21}\,\&\,(L_1\cap R_2)\neq\varnothing\to$ (// **варіанти 18-19**

1. $L_1\cap L_2\neq_T\varnothing\to T_1\cap T_2=[LR]_{22}\cdot LL_{12}++[LR]_{12}\cdot LL_{21}$,

2. $LL_{21}\to T_1\cap T_2=[LR]_{12}\cdot LR_{21}$);

1. $RR_{21}\to T_1\cap T_2=\varnothing$. // **варіант 20**

Алгоритм 2 (розпізнавання варіантів). Усі співвідношення, визначені в алгоритмі 1 - умовні. Умови визначені в термінах відношення часткового порядку "$\leq_T$" і відношення "$\neq_T$". Так, якщо на області T виконується лінійна нерівність типу $R(X)-L(X)\geq_T 0$, це означає, що

$$F_{\max}=\max_{X\in T}(L(X)-R(X))\leq 0\,.$$

Якщо $R(X) \cap L(X) \neq_T \varnothing$, це означає, що

$$\max_{X \in T}(L(X) - R(X)) \geq 0, \ \min_{X \in T}(L(X) - R(X)) \leq 0.$$

З того, що функція $F = L(X) - R(X)$ лінійна і трапецоїд T – опукла багатогранна область, випливає, що максимум і мінімум досягаються в одній з вершин T. Форма представления трапецоїда у вигляді послідовності його проекцій на підпростори все меншої розмірності дозволяє обчислити шуканий максимум за час $O(n^2)$. Покажемо це.

Розглянемо наступну задачу: знайти максимум $F_{\max}$ лінійної функції $F = c_1 x_1 + ... + c_n x_n$ на трапецоїді $T = s_1 \cdot s_2 \cdot ... \cdot s_n$, $s_k = [L_k(X_{k-1}), x_k, R_k(X_{k-1})], k = 1,...,n$. Оскільки розвязок досягається в вершинах трапецоїда T, можуть мати місце три випадки:

1.1. $L_n = -\infty, x_n = -\infty$. Тоді, якщо $c_n > 0$, максимум досягається при $x_n = R_n(X_{n-1})$ (випадок 1.3). Якщо $c_n < 0$, $F_{\max} = +\infty$.

1.2. $x_n = L_n(X_{n-1})$. Тоді задача зводиться до задачі розмірності $n-1$: на трапецоїді $T^{(n-1)} = s_{n-1}....s_1$ знайти максимум $F_{1\max}$ функції $F_1 = c_1 x_1 + ... + c_{n-1} x_{n-1} + c_n L_n(X_{n-1})$.

1.3. $x_n = R_n(X_{n-1})$. Задача зводиться до задачі размірності $n-1$: на трапецоїді $T^{(n-1)} = s_{n-1}....s_1$: знайти максимум $F_{2\max}$ функції $F_2 = c_1 x_1 + ... + c_{n-1} x_{n-1} + c_n R_n(X_{n-1})$.

Таким чином, $F_{\max} = \max(F_{\max 1}, F_{2\max})$.

Мінімум $F = c_1 x_1 + ... + c_n x_n$ обчислюється аналогічно.

Отже, задача розміру n на трапецоїді T зводиться до задачі розміру $n-1$ на трапецоїді $T^{(n-1)}$ за час $O(n)$. Відповідно, задача розміру n вирішується на трапецоїді за час $O(n^2)$.

Алгоритм 3 (перетину полігонів). Операція перетину полігонів $P_1 \bigcap P_2$ описується системою співвідношень - переписуючих правил. Нехай

$P_1 = T_{11} + + T_{12} + + ... + + T_{1k_1}$, $P_2 = T_{21} + + T_{22} + + ... + + T_{2k_2}$.

Представимо P_1, P_2 в вигляді $P_1 = g + G$, $P_2 = h + H$, де

$g = T_{11}, h = T_{21}$, $G = T_{12} + +... + +T_{1k_1}, H = T_{22} + +... + +T_{2k_2}$.

Основне правило має вигляд:

$$(g + +G)(h + +H) = gh + +(gH \cup hG) + +HG \tag{3.5}$$

Нехай $g = T'\cdot[l_g, r_g], h = T''\cdot[l_h, r_h]$. Тоді, при $r_g \le r_h$ - $gH = \varnothing$, а при $r_h \le r_g$ - $hG = \varnothing$.

Це дозволяє спростити правило (3.5). Введемо наступні функції доступу:для $T = T'\cdot s$, $Base(T) \stackrel{df}{=} s$, для $s = [L, x, R]$ $LP(s) = L,\ RP(s) = R$. Тоді (3.4) спрощується:

$$\begin{aligned} RP(Base(g)) \le RP(Base(h)) \to (g + +G)(h + +H) = gh + +hG + +HG \\ else (g + +G)(h + +H) = gh + +gH + +HG \end{aligned} \tag{3.6}$$

Похідні правила виводяться з (3.5) для окремих випадків $G = \varnothing$ або $H = \varnothing$:

$$RP(Base(g)) \le RP(Base(g)) \to g(h + +H) = gh \ \ else\, g(h + +H) = gh + +gH \tag{3.7}$$

$$RP(Base(g)) \le RP(Base(h)) \to (g + +G)h = gh + +hG \ \ else (g + +G)h = gh \tag{3.8}$$

Система переписувань (3.6) - (3.8) представляє алгоритм перетину полігонів.

3.3. Лінійні напівалгебраїчні формули та політрапецоїди.

Визначення 3.7. Нехай $Y = (y_1,..., y_m), x \notin Y,\ T \subseteq Q^m$. Алгеброю x-полісегментів над основою T назвемо конструктивне розширення алгебри x-сегментів, елементи носія якої мають вигляд:

$$S \cdot T = ([L_1, x, R_1] + +[L_2, x, R_2] + +... + +[L_k, x, R_k]) \cdot T,\ L_i = L_i(Y), R_i = R_i(Y), \tag{3.9}$$

причому виконуються умови:

$$L_1 \le_T R_1 \le_T L_2 \le_M R_2 \le_T ... \le_T L_k \le_T R_k,\ R_i \ne L_{i+1}; \tag{3.10}$$

якщо для деякого i $R_i \equiv L_{i+1}$, запис (3.3) скорочується відповідно до (3.2):

$$[L_i, x, R_i] + +[L_{i+1}, x, R_{i+1}] = [L_i, x, R_{i+1}].$$

На алгебрі полісегментів визначено теоретико-множинні операції перетину, об'єднання. Очевидно, що

$$([L_1, x, R_1] + +... + +[L_k, x, R_k]) \cdot T = [L_1, x, R_1] \cdot T \cup ... \cup [L_k, x, R_k] \cdot T.$$

Визначення полісегмента як суми сегментів ілюструє рис.3.1.

$T=[a;b],\ s_1=[L_1(y),x,R_1(y)],\ s_2=[L_2(y),x,R_2(y)],\ s_3=[L_3(y),x,R_3(y)].$

$S=s_1\cdot T++s_2\cdot T++s_3\cdot T=(s_1++s_2++s_3)\cdot T.$

Визначення 3.8. Нехай $X_j=(x_1,...,x_j)$, $j=1,2,...,n$. Визначимо послідовність S_j рекурсивно:

1. $s_{1i}=[a_i,x_1,b_i],a_j,b_j\in Q,i=1,...,k_1.\ S_1=s_{11}++...++s_{1k_1}$ - базовий x_1-полісегмент, $T_1\overset{df}{=}S_1$.

2. Для будь якого $j=1,...,n-1$ S_{j+1}- x_{j+1} - полісегмент над T_j, а $T_{j+1}\overset{df}{=}S_{j+1}\cdot T_j$.

Множину T_n (або просто T) назвемо політрапецоїдом у просторі Q^n. Очевидно, $T=S_n\cdot S_{n-1}\cdot...\cdot S_1$, причому T_j - проекція T_{j+1} на підпростір Q^j, визначена координатами $x_1,...,x_j$.

Політрапецоїд T визначено послідовністю проекцій на спадаючу послідовність підпросторів $Q^n\supset Q^{n-1}\supset...\supset Q^1$, причому кожна проекція - також політрапецоїд.

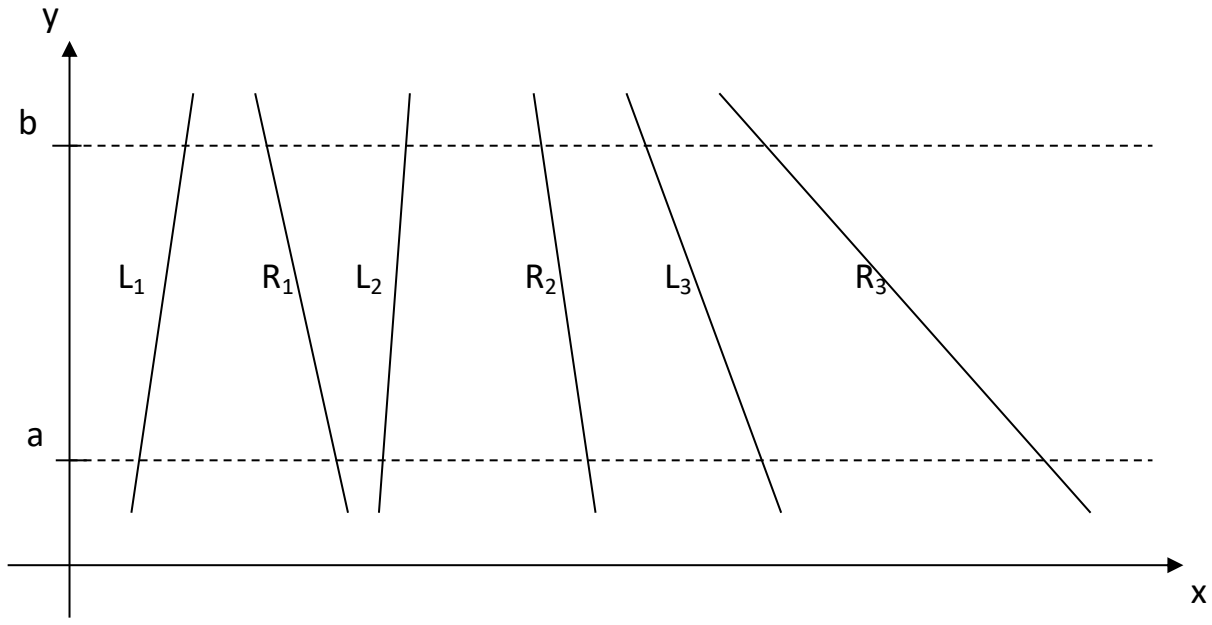


Рис.3.1. *Полісегмент - сума 3-х –сегментів у двовимірному просторі.*

На рис. 3.2. показано політрапецоїд у двовимірному просторі, що складається з $2\times2=4$ трапецій $T_{11},T_{12},T_{21},T_{22}$.

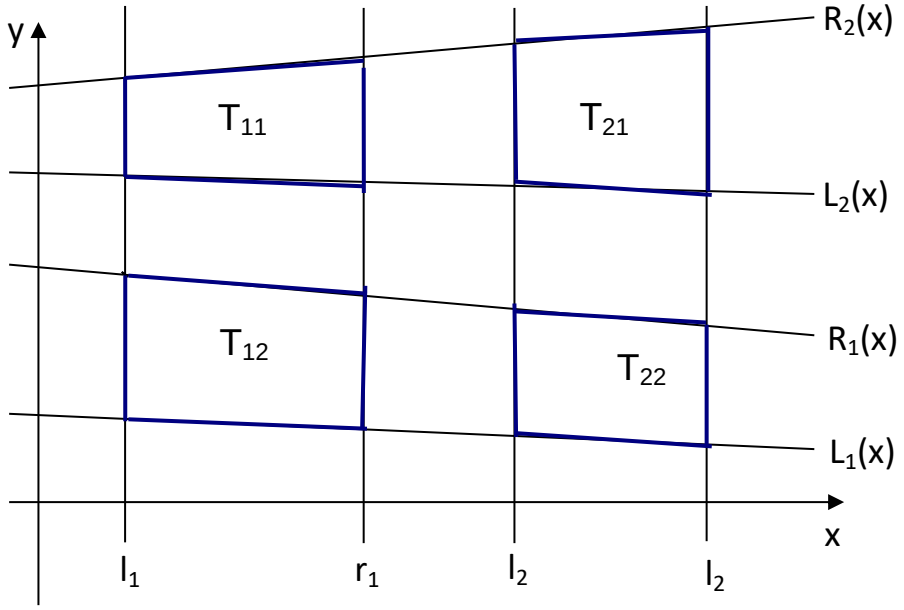


Рис. 3.2. *Політрапецоїд на площині, що складається з* $2\times 2=4$ *трапецій.*

$s_{11}=[R_1(x),y,L_1(x)],\ s_{12}=[R_2(x),y,L_2(x)], s_{21}=[r_1,x,l_1],\ s_{22}=[r_2,x,l_2]$

$S_1=(s_{11}++s_{12}),\ S_2=(s_{21}++s_{22}),\ T=S_1\cdot S_2,$

$T_{11}=s_{11}\cdot s_{21},\ T_{12}=s_{12}\cdot s_{21}, T_{21}=s_{11}\cdot s_{22},\ T_{22}=s_{12}\cdot s_{22}$

$T=T_{11}++T_{12}++T_{21}++T_{22}$

Алгоритм 4 (перетину політрапецоїдів). Нехай PT_1, PT_2 - політрапецоїди і $P=PT_1\cap PT_2$.

Очевидно, P можна представити у вигляді канонічної суми політрапецоїдів. Алгоритм перетину політрапецоїдів обчислює P рекурсивно «знизу-вгору».

Базис рекурсії. Нехай $PT_1=S_{11}\cdot\ldots\cdot S_{1n}$, $PT_2=S_{21}\cdot\ldots\cdot S_{2n}$. Тоді

$$PT_1\cap PT_2=(PT_1^{(n-1)}\cap PT_2^{(n-1)})\cdot(S_{1n}\cap S_{2n}) \qquad (3.11)$$

Співвідношення (3.11) відповідно до визначення 3.6 визначає базис рекурсії - алгоритм обчислення $S_{1n}\cap S_{2n}$, який визначається правилами переписування (3.6) - (3.8) і являє собою суму сегментів числової осі Ox_x, задовольняє (3.10).

Крок рекурсії. Нехай для деякого k $PT_1=S_{11}\cdot\ldots\cdot S_{1n-k}\cdot ST^{(k)}$, $PT_2=S_{21}\cdot\ldots\cdot S_{2n-k}\cdot ST^{(k)}$, де $ST^{(k)}$ - сума політрапецоїдів, що представляє $(S_{1n-k}\cdot\ldots S_{1n})\cap(S_{2n-k}\cdot\ldots S_{2n})$, тобто

$$ST^{(k)}=(S_{1n-k}\cdot\ldots S_{1n})\cap(S_{2n-k}\cdot\ldots S_{2n})\ ,$$

$$PT_1 \cap PT_2 = ((S_{11} \cdot ... \cdot S_{1n-k}) \cap (S_{21} \cdot ... \cdot S_{2n-k})) \cdot ST^{(k)} \quad (3.12)$$

Тоді (3.12) перепишемо у вигляді

$$PT_1 \cap PT_2 = ((S_{11} \cdot ... \cdot S_{1n-k-1}) \cap (S_{21} \cdot ... \cdot S_{2n-k-1}))(S_{1n-k} \cap S_{2n-k}) \cdot ST^{(k)},$$

звівши задачу до кроку рекурсії – обчислення $(S_{1n-k} \cap S_{2n-k}) \cdot ST^{(k)}$.

Оскільки $ST^{(k)}$ являє собою суму політрапецоїдів, $ST^{(k)} = PT_1^{(k)} + +... + +PT_l^{(k)}$,

$(S_{1n-k} \cap S_{2n-k}) \cdot ST^{(k)} = (S_{1n-k} \cap S_{2n-k}) \cdot PT_1^{(k)} + +... + +(S_{1n-k} \cap S_{2n-k}) \cdot PT_l^{(k)}$.

Для обчислення $(S_{1n-k} \cap S_{2n-k}) \cdot PT_j^{(k)}$ застосовуються алгоритми 1-3, описані вище.

Алгоритм 5 (об'єднання політрапецоїдів). Нехай PT_1, PT_2 - політрапецоіди та $P = PT_1 \cup PT_2$. P можна представити у вигляді канонічної суми політрапецоїдів. Нехай

$$PT_1 = PT_1^{(n-1)} \cdot S_{1n},\ PT_2 = PT_2^{(n-1)} \cdot S_{2n}.$$

Припустимо $S_{12n} = S_{1n} \cap S_{2n}, S_{1n}^{'} = S_{1n} - S_{12n},\ S_{2n}^{'} = S_{2n} - S_{12n}$. Тоді $S_{1n} = S_{1n}^{'} + +S_{12n},\ S_{2n} = S_{2n}^{'} + +S_{12n}$ та

$$PT_1 \cup PT_2 = Ord(PT_1^{(n-1)} \cdot S_{1n}^{'} + +PT_2^{(n-1)} \cdot S_{2n}^{1} + +(PT_1^{(n-1)} \cup PT_2^{(n-1)}) \cdot S_{12n} \quad (3.13)$$

де функція $Ord()$ впорядковує складові. Алгоритм обчислення $S_{1n}^{'}, S_{2n}^{'}, S_{12n}$ в (3.13) обчислює S_{12n}, використовуючи алгоритми 1-3 і рекурсивно обчислює $PT_1^{(n-1)} \cup PT_2^{(n-1)}$.

Алгоритм обчислення $S_{1n}^{'} = S_{1n} - S_{12n},\ S_{2n}^{'} = S_{2n} - S_{12n}$ аналогічний алгоритму 3.

Відзначимо, що алгебра лінійних напівалгебраїчних множин побудована як ланцюжок розширень алгебри трапецоїдів, полігонів і політрапецоїдів перевантаженням теоретико-множинних операцій $\&, \vee$.

Теорема 3.2. Кожна лінійна напівалгебраїчна множина M (див. Визначення 3.1) єдиним чином може бути представлена у вигляді суми політрапецоїдів: $M = GT_1 + +GT_2 + +... + +GT_k$.

3.4. Канонічні форми логічних формул над упорядкованими типами даних.

Канонічні форми логічних формул над перелічуваним типом

Перелічувальний тип даних визначимо як скінчено впорядковану множину $A=\{a_1,a_2,...,a_m\}$ з логічними операціями - предикатами порядку і рівності. Обмежимо розгляд предикатом несуворого порядку, оскільки суворий порядок очевидним чином виражається через несуворий порядок, що позначається "$\le,\ge$" і рівність, що позначається "==".

Розглянемо безквантворні формули прикладної логіки предикатів над типом A в сигнатурі логічних зв'язок $<\&,\vee>$ з метою визначення їх канонічних форм, аналогічних канонічним формам лінійних напівалгебраічних формул, розглянутих вище.

Алгоритм обчислення канонічної форми логічної формули над перелічуваних типом.

Нехай $X=\{x_1,...,x_n\}$, $x_1\prec...\prec x_n$ - впорядкована множина предметних змінних і $F(P_1,...,P_k;X)$ - формула прикладної логіки предикатів над A від атомарних предикатів $P_1,...P_k$ і предметних змінних з множини X. Атомарні предикати $P_1,...P_k$ мають вид $x=y,\ x\le y,\ x\ge y$, де $x,\ y$ або предметні змінні з областю визначення A, або елементи A.

Попереднє спрощення $F(P_1,...,P_k;X)$ полягає в елімінації рівності і приведення атомарних нерівностей до виду $x\le y$ або $x\ge y$, де $x\in X$, $y\in X\cup A$, і якщо $y\in X$, то $x\prec y$.

Як і вище, визначимо x-сегменти $[L,x,R]$ як подвійні нерівності $L\le x\le R$, і трапецоїди у вигляді $T=s_1\cdot...\cdot s_n$, $s_i=[L_i,x_i,R_i]$.

Кожну нерівність можна представити у вигляді трапецоїда. Позначимо $\min=a_1,\max=a_m$. Тоді, за визначенням перелічуваного типу, для $x\in X$ *a priori* має місце представлення

$$s_x^{(0)}=[\min,x,\max],\qquad (3.14)$$

а для нерівностей $x \le y, x \ge y$ - відповідно

$$s_{x \le y} = [\min, x, y],\ s_{x \ge y} = [y, x, \max]. \tag{3.15}$$

Відповідно до цього, кожну атомарну нерівність $P_j = x_{j_i} \le y$ або $P_j = x_{j_i} \ge y$ формули $F(P_1,...,P_k; X)$ представимо у вигляді атомарного трапецоїда $T^{(j)} = s_{1j} \cdot ... \cdot s_{nj}$, x_{j_i}-сегмент якого має вигляд (3.15), а інші сегменти - (3.14). Логічні зв'язки $\&, \vee$ замінимо на операції $\cap, \cup$. В результаті, обчислення канонічної форми $F(P_1,...,P_k; X)$ зведено до обчислення значення $F(T^{(1)},...,T^{(k)}; X)$.

Алгоритми обчислення канонічних форм перетину трапецоїдів, полігонів, політрапецоїдів і лінійних напівалгебраїчних формул над перелічуваних типом по суті представлено викладеними вище алгоритмами. Відзначимо тільки, що з огляду на просту праву частину нерівностей алгоритм 2 розпізнавання варіанту має простіший вид, ніж в загальному випадку.

Канонічні форми логічних формул над множинним типом

Множинний тип даних $Set(U)$ визначимо як алгебру підмножин над скінченною множиною – універсумом $U = \{u_1, u_2, ..., u_m\}$, сигнатурою теоретико-множинних операцій $< \cap, \cup, - >$ і операціями приналежності $a \in A$, рівності $a = b$ і заперечення рівності $a \ne b$. Розглянемо безквантворні формули прикладної логіки предикатів над типом $Set(U)$ від змінних $X = (x_1,..., x_n)$ типу Var в сигнатурі логічних зв'язок $< \&, \vee, \neg >$ з метою визначення їх канонічних форм, аналогічних канонічним формам лінійних напівалгебраїчних формул, розглянутих вище.

Попередньо розглянемо кон'юнкцію виду $\Phi = \Phi_0 \,\&\, \Phi_= \,\&\, \Phi_{\ne}$, де

$$\Phi_0 = \begin{cases} x_1 \in A_1 \\ x_2 \in A_2 \\ . \\ x_n \in A_n \end{cases}, \Phi_= = \begin{cases} x_{i1} = y_1 \\ x_{i2} = y_2 \\ . \\ x_{ik} = y_k \end{cases}, \Phi_{\ne} = \begin{cases} x_{j1} \ne z_1 \\ x_{j2} \ne z_2 \\ . \\ x_{jl} \ne z_l \end{cases} \tag{3.16}$$

елементарні кон'юнкти якої мають вигляд відповідно $x \in A,\ x = y,\ x \ne z$, де $A \subseteq U,\ x, y, z \in X$

Можна вважати, що в (3.16) виконані умови:

1. У частину Φ_0 включено всі кон'юнктів виду $x_j = A_j, j = 1,..,n$. (Зокрема, кон'юнкт може мати вигляд $x_j \in U$ або $x = a$, т.е. $x \in \{a\}$).

2. Якщо в $\Phi_=$ є кон'юнкт виду $x = y$, в $\Phi_{\neq}$ відсутній кон'юнкт виду $x \neq y$ (в іншому випадку кон'юнкція помилкова).

Наступне співвідношення можна використовувати для спрощення (3.16):

$$(x \in A) \& (y \in B) \& (x = y) = (x \in A \cap B) \& (x = y) \tag{3.17}$$

Після цього перетворення отримаємо кон'юнкцію виду $\Phi = \Phi_0 \& \Phi_= \& \Phi_{\neq}$

$$\Phi_0 = \begin{cases} x_1 \in A_1 \\ x_2 \in A_2 \\ . \\ x_k \in A_k \end{cases}, \Phi_= = \begin{cases} x_{k+1} = y_1 \\ x_{k+2} = y_2 \\ . \\ x_n = y_{n-k} \end{cases}, \Phi_{\neq} = \begin{cases} x_{j1} \neq z_1 \\ x_{j2} \neq z_2 \\ . \\ x_{jl} \neq z_l \end{cases}, \tag{3.18}$$

в якій $\{y_1,...,y_{n-k}\} \subseteq \{x_1,...x_k\}$. Позначимо $X_0 = (x_1,...,x_k), X_= = (x_{k+1},...,x_n)$.

Отже, якщо розглядати тільки частини $\Phi_0, \Phi_=$ кон'юнкції (3.18), змінні підмножини X_0 можна вважати незалежними, а змінні $X_=$ - пов'язаними співвідношеннями рівностей частини $\Phi_=$ формули (3.18).

Таким чином, канонічна форма $\Phi_0 \& \Phi_=$ формули (3.16) з точністю до порядку елементарних коньюнктів отримана. Єдиність забезпечується упорядкуванням змінних в лівих частинах елементарних кон'юнктів:

$x_1 \succ x_2 \succ ... \succ x_k$; $x_{k+1} \succ x_{k+2} \succ ... \succ x_n$.

Для спрощення $\Phi_0 \& \Phi_{\neq}$ формули (3.18) визначимо відображення $\varphi: X \to X_{base}$:

$$\varphi(x) = \begin{cases} x, если\ x \in X_{base} \\ y, если (x = y) \in F_{equ} \end{cases}$$

Застосуємо до $\Phi_{\neq}$ формули (3.18) співвідношення

$$(x \neq y) = (\varphi(x) \neq \varphi(y)) \tag{3.19}$$

як правило переписування. В результаті в $\Phi_{\neq}$ отримаємо канонізовану систему заперечень рівностей, в якій $\{x_{j1},...,x_{jl}\} \subset X_0$, $\{z_1,...,z_l\} \subset X_0$. Якщо в цій системі існує рівність виду $x \neq x$, формула (3.18) помилкова.

Теорема 3.3. Формула (3.18), отримана як результат перетворень, наведених вище, до формули (3.16), є канонічною формою (3.16).

За аналогією з термінологією, яка використовується вище, канонічну форму (3.18) формули (3.16) назвемо трапецоїдом над множинним типом. За аналогією з поданням трапецоїда над типом Rat, формулу трапецоїда над множинним типом будемо записувати у вигляді

$$\Phi = < \sigma_1 \cdot ... \cdot \sigma_k \cdot \varepsilon_{k+1} \cdot ... \cdot \varepsilon_n, \delta_1 \,\&\, ... \,\&\, \delta_l >,$$

$$\Phi_0 = \sigma_1 \cdot ... \cdot \sigma_k,\ \Phi_{=} = < \varepsilon_{k+1} \cdot ... \cdot \varepsilon_n,\ \Phi_{\neq} = \delta_1 \,\&\, ... \,\&\, \delta_l .$$

Через $\Phi^{(n-1)}$ позначимо трапецоїд $\Phi = < \sigma_1 \cdot ... \cdot \sigma_k \cdot \varepsilon_{k+1} \cdot ... \cdot \varepsilon_{n-1}, \delta_1 \,\&\, ... \,\&\, \delta_l >$. Тоді можна записати $\Phi = \Phi^{(n-1)} \cdot \varepsilon_n$.

Операції перетину та обєднання трапецоїдом над множинним типом

Нехай Φ_1, Φ_2 - трапецоїди над множинним типом. Тоді канонічна форма $\Phi_1 \cap \Phi_2$ обчислюється описаним вище алгоритмом, застосованим до $(\Phi_{0,1} \cdot \Phi_{=,1}) \,\&\, (\Phi_{0,2} \cdot \Phi_{=,2})$.

Алгоритм обчислення $\Phi_1 \cup \Phi_2$ - модифікація алгоритму 5. Нехай $\Phi_1 = \Phi_{0,1} \,\&\, \Phi_{=,1} \,\&\, \Phi_{\neq,1},\ \Phi_2 = \Phi_{0,2} \,\&\, \Phi_{=,2} \,\&\, \Phi_{\neq,2}$. Проекція π_Φ трапецоіда Φ на координатну множину Ox_n змінної x_n дорівнює або A_n, якщо рівність $x_n = A_n$ - елементарний кон'юнкт Φ_0, або A_j, якщо $(x_j = A_j) \in \Phi_0, (x_n = x_j) \in \Phi_{=}$. Нехай $\pi_{\Phi_{1,n}}, \pi_{\Phi_{2,n}}$ - проекції Φ_1, Φ_2 на Ox_n.

Тоді

$$\Phi_1 \cup \Phi_2 = \Sigma_1 + + \Sigma_2 + + \Sigma_3, \tag{3.20}$$

де

$$\pi_{\Sigma_{1,n}} = \pi_{\Phi_{1,n}} - \pi_{\Phi_{2,n}},\ \pi_{\Sigma_3,n} = \pi_{\Phi_{2,n}} - \pi_{\Phi_{1,n}}, \pi_{\Sigma_2,n} = \pi_{\Phi_{1,n}} \cap \pi_{\Phi_{2,n}}, \tag{3.21}$$

$$\Sigma_{0,1}^{(n-1)} = \Phi_{0,1}^{(n-1)},\ \Sigma_{=,1} = \Phi_{=.1},\ \Sigma_{\neq,1} = \Phi_{\neq,1},\ \Sigma_{0,3}^{(n-1)} = \Phi_{0,2}^{(n-1)}, \Sigma_{=,3} = \Phi_{=,2}, \Sigma_{\neq,1} = \Phi_{\neq,3}, \tag{3.22}$$

$$\Sigma_2^{(n-1} = \Phi_1^{(n-1)} \cup \Phi_2^{(n-1)}. \tag{3.23}$$

Як і в алгоритмі 5, формули (3.20) - (3.23) представляють рекурсивний алгоритм обчислення $\Phi_1 \cup \Phi_2$.

3.5. Реалізація алгоритму побудови канонічних форм лінійних напівалгебраїчних формул.

У якості технології реалізації алгоритму побудови канонічної форми лінійної напівалгебраїчної формули ми використовуємо систему інсерційного моделювання IMS [39], створену на основі системи алгебраїчного програмування APS [40] в рамках співпраці між Інституту кібернетики НАН України ім. Глушкова та Херсонським державний університетом під керівництвом академіка НАН України, професора О.А. Летичевського. Як і APS, система IMS є системою для протитипування алгоритмів та моделювання програмних систем.

Інсерційне моделювання, як підхід до дослідження розподілених багатоагентних систем та розробки засобів верифікації розподілених паралельних програм та апаратного забезпечення є одним із основних напрямків досліджень останніх років. Висхідним поняттям інсерційного моделювання є поняття атрибутної транзиційної системи [50], яка, у свою чергу, визначається як пятірка $< S, A, U, T, \varphi >$, де

— S – множина станів,

— A – множина дій, які використовуються для розмітки переходів,

— U – множина атрибутних розміток, які використовуються для розмітки станів,

— T – відношення переходів, $T \subseteq S \times A \times S \cup S \times S$, складається з розмічених переходів $s \xrightarrow{a} s'$ і нерозмічених переходів $s \to s'$. Ця частина структури відповідає звичайному поняттю розміченої транзиційної системи (зі схованими переходами).

Функцією розмітки станів є функція $\varphi: S \to U$, де U – множина $U = D^R$ відображень множини R атрибутів у множину даних D, або у сімейство $U = \left(D_\xi^{R_\xi}\right)\ \xi \in \Xi$ (Ξ - множина типів даних).

У якості інваріантів в інсерційних системах використовуються (в загальному випадку нескінченні) вирази або системи рівнянь в алгебрі поведінок.

Власне сама алгебра поведінок представляє алгебру $< U, A >$, де U – множина поведінок, а A – множина дій. До сигнатури алгебри поведінок входять наступні операції:

1. префіксинг - $a.u$, де a – дія, u – поведінка. Результат операції – нова поведінка,
2. недетермінований вибір - $u + v$ – бінарна операція, визначена на множині поведінок (комутативна, асоціативна та ідемпотентна),

константи:

1. успішне завершення - Δ,
2. невизначена поведінка - $\perp$,
3. тупиковий стан – 0 – нейтральний елемент недетермінованого вибору

та бінарне відношення апроксимації - $\subseteq$ - відношення часткового порядку з найменшим елементом $\perp$.

Операції сигнатури алгебри поведінок є монотонними та безперервними відносно відношення апроксимації.

Основну роль відіграє повна алгебра поведінок $F(A)$, яка містить межі всіх спрямованих множин і, отже, в ній має місце теорема про найменшу нерухому точку. Точну конструкцію алгебри $F(A)$ (для довільної, в тому числі і нескінченної множини дій) викладено в [41].

Спроєктуємо описані вище алгебри в термінах розширень (подвійна стрілка) і наслідування (жирна стрілка) (рис. 3.3) [42-44].

LinMonom – одномірний векторний простір над полем Coef, $\Sigma = \{++, --, *, /, <, >, ==\}$ (ділення і множення на число) і носієм $\mathbf{S} = \{c * v, v \in Variable, c \in Coef, c \neq 0\}$.

LineSpace – векторний простір лінійних комбінацій над полем Coef, $\Sigma = \{++, --, *, /, ==\}$ (ділення і множення на число) і носієм $\mathbf{S} = \{lm + ls, lm \in LinMonom, ls \in LineSpace\}$.

Affspace – афінний простір лінійних комбінацій з вільним членом над полем Coef, $\Sigma = \{++, -, *, /, NOD, NOK <, >, ==\}$ (ділення і множення на число) і носієм $\mathbf{S} = \{ls + +c, ls \in LinSpace, c \in Coef\}$.

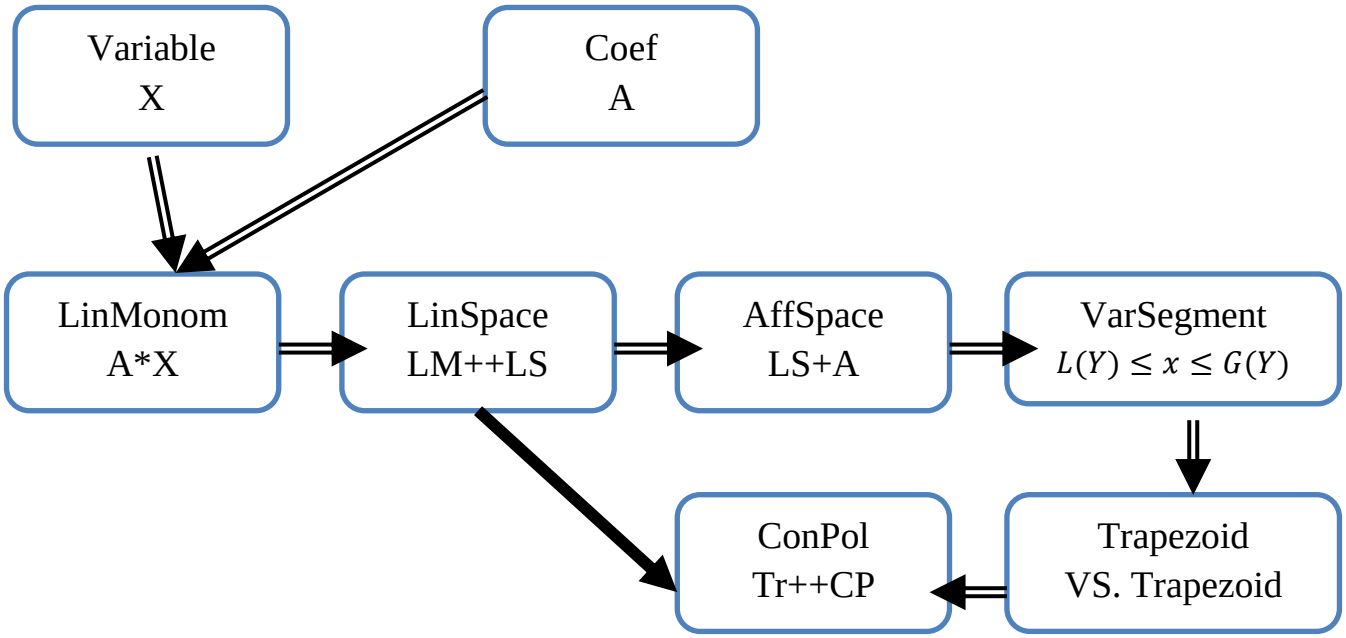


Рис. 3.3. *Діаграма розширень і наслідування сортів*

Розширення алгебри відбувається за рахунок введення нових конструкторів елементів носіїв, описаних вище, операція перетину трапецоїдов визначає спадкування між LinSpace і ConPol.

Для реалізації такого алгоритму засобами інсерційного моделювання необхідно реалізувати функцію розгортання станів (unfolding) і функцію занурення агентів в середовище (insertion) [41]. Обчислення в алгебрі LinSpace легко реалізувати за допомогою оператора мови APLAN mrg [44] (операція «++» визначається аналогічно «+»), тому в подальшому будемо вважати, що в системі лінійних нерівностей всі нерівності приведено до вигляду: $f_k(x_n, \ldots x_1) \le 0$.

В якості початкового стану візьмемо всю область допустимих рішень. Для цього знайдемо список всіх змінних системи нерівностей і приведемо цю область до носія алгебри **Trapezoid**: $(-\infty, x_n, +\infty).(-\infty, x_{n-1}, +\infty)$..... Квадратні дужки будемо використовувати тільки для того, щоб позначити входження кінця відрізка в проміжок.

Функція розгортання станів для цього прикладу визначається як конструктор сорту **VarSegment**:

$unfold\_rs := rs(x, y)($
$(x \le 0) = SolveLinUnEq(x \le 0),$
$(x < 0) = SolveLinUnEq(x < 0)$
);

Функція $SolveLinUnEq$ реалізація конструктора сорту **LinUnEqu**.

Функція занурення містить послідовний розгляд кожного обмеження і операцію рекурсивного перетину обмежень з трапецоїдом:

$ins := rs(E, P, x, y)($

$$E[Delta] = Delta, \tag{3.24}$$

$$E[x \,\&\, y] = ins(ins\, E[x])[y], \tag{3.25}$$

$$(x + + y)[P] = ins\, x[P] + + ins\, y[P], \tag{3.26}$$

$$(x.y)[P] = split\_tr(ins\, x[P].ins\, y[P]), \tag{3.27}$$

$$E[(x < y)] = cross\_vs(x < y, E, P), \tag{3.28}$$

$$E[(x \le y)] = cross\_vs(x \le y, E, P), \tag{3.29}$$

$$E[P] = E[\text{short\_intens}\, P] \tag{3.30}$$

);

Правило (3.24) означає, що занурення термінального стану Delta закінчується формуванням траси «успішне завершення».

(3.25) - реалізація послідовного розгляду нерівностей.

(3.26), (3.27) - перетин обмеження з опуклим многогранником і трапецоїдом відповідно.

(3.28), (3.29) - перетин обмеження з елементом сорту VarSegment функція *cross_vs*.

(3.30) - виклик функції розгортання станів. Функція split_tr реалізує закон дистрибутивності: $x.(y + + z) = x.y + + x.z$.

Для вхідних даних програми використовується терм: $1[L_1 \& L_2 \ldots \& L_n]$ (L_j - лінійна нерівність), тоді функція *cross_vs* при $E = 1$ приводить стан E до носія сорту **Trapezoid**.

Зважаючи на те, що більшість систем доведення та спрощувачів використовують розроблені на мові **C++** модулі, або, власне, реалізовані на мові C, C++, аналогічну реалізацію алгоритму побудови канонічної форми напівалгебраїчних формул було виконано і засобами C++ [49].

Основними функціями, що відображають роботу алгоритму є функція node_ptr build_concrete_values (Додаток 3) – для побудови довільної формули (включаючи функціонали, тощо) та функція node_ptr makeCanForm (Додаток 4) – алгоритм побудови канонічної форми.

Запис формули для спрощення виконується безпосередньо у файлі main.cpp – функція test1.

```
void test1(fpl_ptr &fpl){
 attrs.clear();
 add_simple_attr(fpl,"x","int");
 intr_ptr ii(*fpl);
 node_ptr order = get_term_string (ii,char_ptr("x"));
 node_ptr t = get_term_string
(ii,char_ptr("x>0&x>7&x<12"));
 node_ptr res = build_concrete_values (fpl,t,order);
 std::cout<<"Test1:"<<fpl->printNodeInBuf(res)<<std::endl;
}
```

Main-функція програми виглядає наступним чином.

```
int main(int argc,const char *argv[]){
 void init_fpl(fpl_ptr &_clewptr);
 fpl_ptr fpl;
 fpl.New();
 init_fpl(fpl);
 test1(fpl);
  return 0;
}
```

3.6. Модифікація алгоритму побудови канонічної форми лінійних напівалгебраїчних формул. Застосування алгоритму поповнення.

Алгоритм поповнення широко застосовується у конструктивній теорії поліноміальних ідеалів. Автором алгоритму є Б.Бухбергер [45, 46], який застосував його для побудови базису ідеалу кільця многочленів (відомого як базис Гребнера), що володіє «хорошими» властивостями. Якщо базис Гребнера ідеалу побудовано, багато класичних задач теорії поліноміальних ідеалів можуть бути вирішені ефективно. Це, наприклад, задача приналежності, задача порівняння, та інші задачі. Фактично метод базисів Гребнера став основним методом конструктивної теорії поліноміальних ідеалів.

Алгоритм поповнення, відомий як алгоритм Кнута-Бендикса [47] використовується в теорії напівгруп з визначаючими співвідношеннями, а також в теорії систем переписування термів [48].

Коротко сформулюємо основну задачу.

Нехай $X = (x_1, x_2, ..., x_n)$ - набір змінних і $L_1(X), L_2(X), ..., L_m(X)$ - набір лінійних неоднорідних нерівностей. Допускаються як суворі, так і несуворі лінійні нерівності. В n - вимірному евклідовому векторному просторі, координатні осі якого іменовані змінними набору X, кожна лінійна нерівність інтерпретована як замкнутий або відкритий півпростір. Таким чином, можна визначити алгебру множин з базисом $L_1(X), L_2(X), ..., L_m(X)$ та теоретико-множинною сигнатурою $\langle \&, \vee, \_ - \rangle$. Ця алгебра нпазивается алгеброю лінійних напівалгебраїчних множин, Формули - елементи цієї алгебри називаються лінійними напівалгебраїчними формулами. Відповідна алгебра множин - інтерпретація алгебри формул на $W^n(X)$ - алгебра лінійних напівалгебраїчних множеств. В задачі потрібно побудувати канонічну форму для формул цієї алгебри.

У [38] викладено так званий метод трапецоїдів, що вирішує завдання побудови канонічної форми системи строгих лінійних нерівностей нескладними перетвореннями. У розділі 3 викладено досить складний алгоритм вирішення

загальної задачі побудови канонічної форми лінійних напівалгебраїчних формул. Складність полягає в алгоритмі реалізації операції об'єднання в даній алгебрі.

Метод поповнення дозволяє просто вирішити дану задачу. Саме, нехай $f(L_1, L_2, ..., L_m)$ - лінійна напівалгебраїчна формула, канонічну форму якої потрібно побудувати.

Побудуємо стандартний базис $\{B_1, B_2, ..., B_l\}$ системи множин $\mathrm{L_1(X)}, \mathrm{L_2}(X), ..., \mathrm{L_m}(X)$ використавши спеціальний алгоритм побудови, наведений нижче.

Алгоритм побудови стандартного базису.

1. Нехай ми маємо систему множин $A = \{A_1, A_2, ..., A_m\}$. Доповнимо систему універсумом U: $\overline{A} = \{U, A_1, A_2, ..., A_m\}$.

2. $\overline{B}^{(0)} = \{U\}$

3. Нехай $B^{(j)} = \{B_1^{(j)}, B_2^{(j)}, ..., B_k^{(j)}\}$ - стандартний базис $\overline{A}^{(j)} = \{U, A_1, A_2, ..., A_j\}$, тоді для всіх пар виду $(B_i^{(j)}, A_{j+1}), i = 1,2,...,k$ виконуються перетворення:

$$B_i^{(j)} \cap A_{j+1} = \varnothing \to B_i^{(j+1)} = B_i^{(j)} \quad // \quad B_i^{(j)} \cap \overline{A}_{j+1} = B_i^{(j)}$$

$$B_i^{(j)} \cap \overline{A}_{j+1} = \varnothing \to B_i^{(j+1)} = A_{j+1} \quad // \quad B_i^{(j)} \cap A_{j+1} = A_{j+1}$$

$$B_{i,1}^{(j+1)} = B_i^{(j)} A_{j+1}, B_{i,2}^{(j+1)} = B_i^{(j)} \overline{A}_{j+1} \quad // \quad B_i^{(j)} \cap \overline{A}_{j+1} \neq \varnothing, B_i^{(j)} \cap A_{j+1} \neq \varnothing$$

Таким чином, кожен елемент стандартного базису $\{B_1, B_2, ..., B_l\}$ системи множин $\mathrm{L_1(X)}, \mathrm{L_2}(X), ..., \mathrm{L_m}(X)$ має вигляд $B_j = L_{j1}^{\alpha_{j1}} \,\&\, L_{j2}^{\alpha_{j2}} \,\&\, ... \,\&\, L_{jn}^{\alpha_{jn}}$, де

$$L^{\alpha} = \begin{cases} L, \alpha = 0 \\ \neg L, \alpha = 1 \end{cases}$$

Опишемо тепер алгоритм побудови канонічної форми F. Канонічна форма має вид $Can(F) = B_{i1} \vee B_{i2} \vee ... \vee B_{ik}$. Тому після побудови базису потрібно визначити, які з базисних елементів належать $\mathrm{Can(F)}$. Але неважко показати, що якщо $B = L_1^{\alpha_1} \,\&\, L_2^{\alpha_2} ... L_n^{\alpha_n}$ то $B \in Can(F) \Leftrightarrow F(\alpha_1, \alpha_2, ... \alpha_n) = 1$

Тому, після того, як базис $\{B_1, B_2, ..., B_l\}$ побудовано, для кожного базисного елемента потрібно просто обчислити відповідне значення $F(\alpha_1, \alpha_2, ...\alpha_n)$. Залишилося відмітити, що побудова непорожнього перетину $L_1^{\alpha_1} \& L_2^{\alpha_2} ... L_n^{\alpha_n}$ здійснюється невеликою модифікацією алгоритму методу трапецоїдів із [37].

Висновки до розділу 3

Результати аналізу функціональних особливостей існуючих систем побудови доказів (підрозділ 3.1) (на прикладі CVC4, MathSAT5, QEPCAD, Singular, COCOA, MiniZinc, STP, RedLog, Satallax, Isabelle, E-SETHEO, Minisat, SMTInterpol, TPS / ETPS, Paradox, Gandalf, Vampire) дозволяють зробити висновок про відсутність системи, яка повністю задовольняє поставлені у дослідженні вимоги. Зокрема, особливо актуальною, насьогодні, залишається задача реалізації відповідних алгоритмів, що дозволяють виконувати спрощення формул на більш ефективному рівні.

Відповідно до завдань дисертаційного дослідження, у Розділі 3 відображено наступні результати:

1) Описано результати аналізу існуючих систем побудови доказів, зокрема інструментів спрощення логічних формул.

2) У підрозділах 3.2 – 3.4 запропоновано алгоритм побудови канонічної форми лінійних півалгебраїчних (ЛПФ) формул. Зокрема, описано алгоритми побудови канонічних форм логічних формул над перелічуваним та множиним типами.

Основний результатом є визначення канонічної форми ЛПФ, що володіє властивістю єдиності і іншими корисними властивостями, а також опис алгоритму її побудови. Пропонована канонічна форма ЛПФ - пряме узагальнення канонічної форми представлення системи лінійних нерівностей і алгоритму її побудови, наведених в [37, 38].

3) У підрозділі 3.5. наведено реалізацію алгоритму побудови канонічної форми ЛПФ засобами IMS та C++.

4) У підрозділі 3.6. запропоновано модифікацію алгоритму побудови канонічної форми лінійних напівалгебраїчних формул за рахунок застосування алгоритму поповнення.

Результати дослідження апробовано на Міжнародній науковій конференції Теоретичні та прикладні аспекти побудови програмних систем, м. Київ, 5-9 грудня 2016 р. [51], Міжнародній науковій конференції ICTERI, м. Київ, 15-19 травня 2017 р. [52], викладено у науковій публікації у міжнародниому науково-теоретичному журналі «Кібернетика і системний аналіз» [53].

Список літератури до Розділу 3

1. ACL2 Version 8.3 [Електронний ресурс]. – Режим доступу до ресурсу: http://www.cs.utexas.edu/~moore/acl2/.

2. Kaufmann M., Moore J. S. Design Goals for ACL2. / M. Kaufmann, J. S. Moore. // Proc. of 3-rd International School and Symposium on Formal Techniques in Real Time and Fault Tolerant Systems. - 1994. - P. 92-117.

3. E-SETHEO [Електронний ресурс]. – Режим доступу до ресурсу: http://www4.informatik.tu-muenchen.de/~schulz/WORK/e-setheo.html.

4. The E Theorem Prover [Електронний ресурс]. – Режим доступу до ресурсу: http://www.eprover.org/.

5. Bachmair L., Ganzinger H. Rewrite-Based Equational Theorem Proving with Selection and Simplification. / L. Bachmair, H. Ganzinger. // Journal of Logic and Computation. - 1994. - №4(3). – P. 217 - 247.

6. The KeY to Software Correctness [Електронний ресурс]. – Режим доступу до ресурсу: http://www.key-project.org/.

7. Beckert B. Verification of Object-Oriented Software. The KeY Approach / B. Beckert, R. Hähnle, P. Schmitt., 2007. – 658 с. – (4334).

8. Riazanov A., Voronkov A. The Design and Implementation of Vampire. / A. Riazanov, A. Voronkov // AI Communications. – 2002. - №15(2). – P. 91 - 110.

9. Waldmeister [Електронний ресурс]. – Режим доступу до ресурсу: http://www.mpi-inf.mpg.de/~hillen/waldmeister/.

10. Darwin [Електронний ресурс]. – Режим доступу до ресурсу: ttp://combination.cs.uiowa.edu/Darwin/.

11. PVS [Електронний ресурс]. – Режим доступу до ресурсу: http://pvs.csl.sri.com/

12. HOL [Електронний ресурс]. – Режим доступу до ресурсу: https://hol-theorem-prover.org/

13. Isabelle [Електронний ресурс]. – Режим доступу до ресурсу: https://isabelle.in.tum.de/

14. Coq [Електронний ресурс]. – Режим доступу до ресурсу: https://coq.inria.fr/

15. CVC4 [Електронний ресурс]. – Режим доступу до ресурсу: https://cvc4.github.io/

16. MathSAT5 [Електронний ресурс]. – Режим доступу до ресурсу: https://mathsat.fbk.eu/

17. QEPCAD [Електронний ресурс]. – Режим доступу до ресурсу: https://www.usna.edu/CS/qepcadweb/B/QEPCAD.html

18. Singular W. Decker, G.-M. Greuel, G. Pfister and H. Sch¨onemann. Singular 4- 0-2 — A computer algebra system for polynomial computations. [Електронний ресурс]. – Режим доступу до ресурсу: http://www.singular.uni-kl.de

19. COCOA [Електронний ресурс]. – Режим доступу до ресурсу: http://cocoa.dima.unige.it/

20. CoCoALib [Електронний ресурс]. – Режим доступу до ресурсу: http://cocoa.dima.unige.it/cocoalib/

21. FlatZinc [Електронний ресурс]. – Режим доступу до ресурсу: https://www.gecode.org/flatzinc.html

22. STP [Електронний ресурс]. – Режим доступу до ресурсу: https://stp.github.io/

23. RedLog [Електронний ресурс]. – Режим доступу до ресурсу: https://www.redlog.eu/

24. Satallax [Електронний ресурс]. – Режим доступу до ресурсу: https://www.ps.uni-saarland.de/~cebrown/satallax/

25. Gernot S., Andreas W. E-SETHEO: Design, configuration and use of a parallel automated theorem prover / S. Gernot; W. Andreas. // Australasian Joint Conference on Artificial Intelligence. Springer, Berlin, Heidelberg. - 1999. - P. 231-243.

26. Niklas S., Niklas E. Minisat v1. 13-a sat solver with conflict-clause minimization. / S. Niklas, E. Niklas // SAT. – 2005. - №53. – P. 1 - 2.

27. Jürgen C. SMTInterpol: An interpolating SMT solver / C. Jürgen, H. Jochen, N. Alexander // International SPIN Workshop on Model Checking of Software. Springer, Berlin, Heidelberg. - 2012. - P. 248-254.

28. TPS: A hybrid automatic-interactive system for developing proofs / A. B. Peter, B. E. Chad // Journal of Applied Logic. – 2006. - №4(4). – P. 367-395.

29. Paradox 3.0., https://web.archive.org/web/20070108005818/ http://www.cs.chalmers.se/~koen/folkung/

30. Tammet T. Gandalf version c-1.0c Reference Manual / T. Tammet., 1997.

31. Vampire [Електронний ресурс]. – Режим доступу до ресурсу: http://www.vprover.org/

32. Kovács L., Voronkov A. First-order theorem proving and Vampire. / L. Kovács, A. Voronkov // International Conference on Computer Aided Verification. Springer, Berlin, Heidelberg. - 2013. - P. 1-35.

33. Fourier–Motzkin elimination [Електронний ресурс]. – Режим доступу до ресурсу: http://en.wikipedia.org/wiki/Fourier%E2%80%93Motzkin_elimination

34. Collins G. E. Quantifier elimination by cylindrical algebraic decomposition — twenty years of progress / G. E. Collins // Quantifier elimination and cylindrical algebraic decomposition (Ed. B. F. Caviness and J. R. Johnson). — New York: Springer–Verlag, 1998. — P. 8–23.

35. Omega Test [Електронний ресурс]. – Режим доступу до ресурсу: http://www.umiacs.umd.edu/~hismail/Omega/node3.html

36. Badros G. The Cassowary linear arithmetic constraint solving algorithm / G. Badros, A. Borning, P. Stuckey // ACM transactions on computer-human interaction (TOCHI). — 2001. — №8(4) — P. 267–306.

37. Lvov M. S., Peschanenko V. S. Trapezoid method for solving systems of linear inequalities and its implementation in insertion modeling / M. S.Lvov, V. S. Peschanenko //Cybernetics and Systems Analysis. – 2012. – T. 48. – №. 6. – C. 931-942

38. Lvov M. S. Algebraic approach to the problem of solving systems of linear inequalities / M. S.Lvov // Cybernetics and Systems Analysis. – 2010. – T. 46. – №. 2. – C. 326-338.

39. IMS [Електронний ресурс]. – Режим доступу до ресурсу: http://www.apsystems.org.ua/node/7

40. Letichevsky A., Letichevsky A. (jr), Peschanenko V. APS and Tools / A. Letichevsky, A. Letichevsky (jr), V. Peschanenko // Herald of the V.N.Karazin Kherkiv Nitional University. – 2010. - p. 165-173.

41. Letichevsky A. Algebra of behavior transformations and its applications / A. Letichevsky // Structural theory of Automata, Semigroups, and Universal Algebra, NATO Science Series II. Mathematics, Physics and Chemistry. – 2005. – № 207. – P. 241–272.

42. Львов М.С. Синтез інтерпретаторів алгебраїчних операцій в розширеннях багатосортних алгебр / М.С.Львов // Вісник Харьківського національного университету. - 2009.-№847.-С.221-238. - (Серія «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління»).

43. Львов М.С. Метод спадкування при реалізації алгебраїчних обчислень в математичних системах навчального призначення / М.С.Львов// Системи управління, навігації та зв'язку. - К: ЦНДІ НіУ, 2009.- Вип.3 (11).- С.120-130.

44. Песчаненко В.С. Об одном подходе к проєктированию алгебраических типов данных / В.С. Песчаненко // Проблеми програмування. – 2006. – №2 – 3. – С. 626 – 634.

45. . Buchberger B. Basic features and development of the Critical-pair/Completion procedure / B. Buchberger // Lect. Notes Comput. Sci. — 1985. — Vol. 202. — P. 1—45.

46. Бухбергер Б., Лоос Р. Упрощение алгебраических выражений / Б. Бухбергер, Р. Лоос // Компьютерная алгебра. Символьные и алгебраические вычисления. — М.: Мир. 1986. — С. 23—65.

47. Knuth D.E., Bendix P.B. Simple Word Problems in Universal Algebra / D.E. Knuth, P.B. Bendix // Automation of Reasoning. – 1983. – P. - 342–376.

48. Book R. V., Otto F. String-rewriting systems. / R. V. Book, F. Otto // String-Rewriting Systems. Springer, New York, NY. - 1993. - p. 35-64.

49. Реалізація алгоритму побудови КФ ЛНПФ [Електронний ресурс]. – Режим доступу до ресурсу: https://drive.google.com/drive/folders/1jaK0S5rTI75ZX5qUEUNbQ8I3LdT1W9JY?usp=sharing

50. Park D. Concurrency and automata on infinite sequences / D. Park // Theoretical computer science. Springer, Berlin, Heidelberg. - 1981. - P. 167-183.

51. The canonical forms of logical formulas off the data types / M. Lvov, V. Peschanenko, O. Letychevskyi, Yu. Tarasich // 13th International Conference Theoretical and Applied Aspect of Program Systems Development. – 2016.

52. The canonical forms of logical formulae over the data types and their using in programs verification / Lvov, M., Peschanenko, V., Letychevskyi, O., Tarasich, Y. // CEUR Workshop Proceedings. – 2017. - №1844. - P. 536–554.

53. Algorithm and Tools for Constructing Canonical Forms of Linear Semi-Algebraic Formulas / [M. Lvov, V. Peschanenko, Y. Tarasich та ін.]. // Cybernetics and Systems Analysis. – 2018. – №54. – С. 993–1002.

ВИСНОВКИ

Проблема створення надійного програмного забезпечення спеціалізованих обчислювальних систем є загальновідомою актуальною проблемою. Одним із основних підходів до розв'язання цієї проблеми є використання методів верифікації програмного забезпечення, що спираються, зокрема, на статичний аналіз програмного забезпечення. Насьогодні, запропоновано безліч підходів та створено багато як експериментальних, так і комерційних спеціалізованих програмних систем статичного аналізу та, з його використанням, верифікації програм (аналіз методів та систем статичного аналізу наведено у підрозділі 1.2 розділу 1, аналіз функціональних особливостей систем доведення (пруверів) та результати їх випробування приведено у підрозділі 3.1 розділу 3). Основними недоліками цих систем є по-перше, використання обмежених типів програмних інваріантів, по-друге, використання різних математичних моделей об'єктних програм, по-третє, вузька спеціалізація цих систем. Пошук так званих програмних інваріантів залишається насьогодні відкритою проблемою розробки інструментів та методів статичного аналізу.

Проблемі генерації інваріантів програм присвячено Розділ 2.

У підрозділі 2.1 Розділу 2 приведено поняття власного полінома лінійного оператора та алгоритм побудови власних поліномів, визначено зв'язок між власними поліномами лінійних операторів та поліноміальними інваріантами лінійних циклів програм.

У підрозділі 2.2. Розділу 2 представлено новий метод доказу інваріантності системи лінійних нерівностей, а також завершуваності лінійно визначених ітеративних циклів імперативних програм.

Запропоновані методи доказу та генерації інваріантних рівностей та доказу інваріантних нерівностей може бути покладено в основу загального алгоритму доказу інваріантності системи лінійних нерівностей та поліноміальних ріностей для лінійно-визначених програм.

Ще однією відкритою проблемою статичних методів доведення властивостей моделей є робота з досить складними формулами (проблема

побудови канонічних і нормальних форм в символьному моделюванні), складність яких може підвищуватися крок за кроком (побудова інваріантів і т.д.).

Новий алгоритм побудови канонічної форми лінійних півалгебраїчних (ЛПФ) формул, включаючи алгоритми побудови канонічних форм логічних формул над перелічуваним та множиним типами запропоновано у Розділі 3 (підрозділи 3.1 – 3.4). Побудована канонічна форма ЛПФ володіє властивістю єдиності і іншими корисними властивостями. У підрозділі 3.6 цього ж розділу запропоновано модифікацію алгоритму побудови канонічної форми лінійних напівалгебраїчних формул за рахунок застосування алгоритму поповнення.

Реалізацію алгоритму побудови канонічної форми ЛПФ засобами IMS та C++ приведеноу підрозділі 3.5.

Відповідно до поставлених завдань дослідження у дисертаційній роботі:

1. Проведено дослідження існуючих спеціалізованих систем статичного аналізу програм, а також алгоритмів комп’ютерної алгебри та інсерційного моделювання в системах статичного аналізу та верифікації програмного забезпечення.
2. Проведено аналіз існуючих алгоритмів побудови канонічних форм логічних формул та інструментів спрощення (солверів).
3. Уперше представлено новий алгоритм побудови канонічної форми лінійних напівалгебраїчних формул, що володіє властивістю єдиності та іншими корисними властивостями, а також алгоритм її побудови.
4. Уперше реалізовано алгоритм побудови канонічної форми логічних формул засобами інсерційного моделювання (система інсерційного моделювання IMS) та C++.
5. Представлено новий метод доказу інваріантності системи лінійних нерівностей і завершення деяких лінійних ітераційних циклів імперативних програм; розглянуто та наведено алгоритм обчислення основних інваріантів лінійного оператора жорданової клітинки та алгоритм обчислення основних інваріантів діагоаналізуємого лінійного оператора з непривідним мінімальним характеристичним поліномом.

Практичне значення наукових результатів полягає в можливості застосування наукових положень і висновків дослідження в задачах верифікації програмного забезпечення, а саме – в розробці спеціалізованих програмних систем верифікації формальних моделей програм. На теперішньому етапі виконується дослідження застосування методів формальної верифікації, зокрема методів статичного аналізу, та використання систем верифікації для верифікації формальних моделей програм, економічних, біологічних процесів, нормативно-правових документів, тощо, продовжується робота над дослідженням математичних методів та розробкою ефективних алгоритмів комп'ютерної алгебри статичного аналізу програм. Відповідні результати представлено у [218-220].

Розвиток даного підходу у верифікації матиме значний вплив на ефективність та надійність програмного забезпечення для медичної, військової, економічної, авіаційної та ін. галузей.

Результати дослідження також імплементовано у навчальні курси, що викладаються для студентів ІТ спеціальностей (Якість ПЗ та тестування, Формальні методи ферифікації ПЗ, Інженерія програмного забезпечення, тощо).

Список використаних джерел

1. 2016 Cyber Crime Study & the Risk of Business Innovation. Ponemon Institute. [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: www.ponemon.org.

2. Microsoft Security Development Lifecycle [Електронний ресурс] – Режим доступу до ресурсу: https://www.microsoft.com/en-us/securityengineering/sdl

3. ГОСТ Р 56939-2016 «Захист інформації. Розробка безпечного програмного забезпечення. Загальні вимоги» [Електронний ресурс] – Режим доступу до ресурсу: http://docs.cntd.ru/document/1200135525

4. IEEE STANDARDS ASSOCIATION, et al. 829-1998 IEEE Standard for Software Test Documentation. Technical report, 1998.

5. IEEE 1008-1987. Standard for Software Unit Testing. In IEEE Standards: Software Engineering, Volume Two: Process Standards. New York: IEEE, 1999.

6. ISO/IEC 14598-1 Information technology — Software product evaluation — Part 1: General overview. Geneva, Switzerland: ISO, 1999.

7. ISO/IEC 14598-2 Information technology — Software product evaluation — Part 2: Planning and management. Geneva, Switzerland: ISO, 2000.

8. ISO/IEC 14598-3 Information technology — Software product evaluation — Part 3: Process for developers. Geneva, Switzerland: ISO, 2000.

9. ISO/IEC 14598-4 Information technology — Software product evaluation — Part 4: Process for acquirers. Geneva, Switzerland: ISO, 1999.

10. ISO/IEC 14598-5 Information technology — Software product evaluation — Part 5: Process for evaluators. Geneva, Switzerland: ISO, 1998.

11. ISO/IEC 14598-6 Information technology — Software product evaluation — Part 6: Documentation of evaluation modules. Geneva, Switzerland: ISO, 2001.

12. ISO/IEC 15504-2 Information technology — Process assessment — Part 2: Performing an assessment. Geneva, Switzerland: ISO, 2003.

13. ISO/IEC 15504-3 Information technology — Process assessment — Part 3: Guidance on performing an assessment. Geneva, Switzerland: ISO, 2004.

14. ISO/IEC 15504-4 Information technology — Process assessment — Part 4: Guidance on use for process improvement and process capability determination. Geneva, Switzerland: ISO, 2004.

15. ISO/IEC 15504-5 Information technology — Process assessment — Part 5: An exemplar Process Assessment Model. Geneva, Switzerland: ISO, 2006.

16. IEEE 1028 Standard for Software Reviews. New York: IEEE, 1998.

17. 1012-2004 - IEEE Standard for Software Verification and Validation [Електронний ресурс] – Режим доступу до ресурсу: https://standards.ieee.org/findstds/standard/1012004.htmlhttps://standards.ieee.org/findstds/standard/1012-2004.html.

18. 1012-2012 - IEEE Standard for System and Software Verification and Validation [Електронний ресурс] – Режим доступу до ресурсу: https://standards.ieee.org/findstds/standard/1012-2012.html.

19. Nelson M. Curing the Software Requirements And Cost Estimating Blues / M. Nelson, J. Clark, M. Spurlock. // Program manager. – 1999. – №28. – С. 54–65.

20. Rashinkara P. System-on-a-Chip Verification: Methodology and Techniques / P. Rashinkara, P. Paterson, L. Singh.. – 392 с.

21. Boehm B., Basili V. Software Defect Reduc-tion Top 10 List // IEEE Computer. – 2001. – Vol. 34(1). – P. 135–137.

22. Hoare T. The verifying compiler: A grand challenge for computing research / Tony Hoare // International Conference on Compiler Construction. Springer, Berlin, Heidelberg. – 2003. – С. 262–272.

23. О безошибочных программах [Електронний ресурс] // Открытые системы. – 2004. – Режим доступу до ресурсу: https://www.osp.ru/os/2004/07/185003.

24. Вудкок Д. Первые шаги к решению проблемы верификации программ / Джим Вудкок. // Открытые системы. – 2006. – №8.

25. Сертификация и тестирование программного обеспечения [Електронний ресурс] – Режим доступу до ресурсу: https://npo-echelon.ru/services/certification/

26. Статический анализ безопасности кода // Программная инженерия и информационная безопасность. – 2013. - № 1. - с. 50.

27. Безпека смарт-контрактів [Електронний ресурс] – Режим доступу до ресурсу: https://xakep.ru/2018/09/19/ethereum-top10-exploits/

28. Clarke E. A tool for checking ANSI-C programs / E. Clarke, D. Kroening, F. Lerda // Lecture Notes in Computer Science. – 2004. – № 2988. – P. 168–176.

29. Ершов А.П. Введение в теоретическое программирование: беседы о методе. / А.П. Ершов. – М.: Наука, 1977. – 288 с.

30. Основи дискретної математики / [Ю. Капітонова, С. Кривий, О. Летичевський та ін.]. – Київ: Наукова думка, 2002. – 580 с.

31. Львов М. С. О реализации вычислений в задачах анализа программ, определенных над векторными пространствами / М. С. Львов. // Проблемы программирования. – 2004. – №2. – С. 95–101.

32. Львов С. М. О технологиях построения и обработки математических моделей программ / С. М. Львов // Проблемы программирования. – 2007. - №3. – С. 41-48.

33. Львов С. М. Инвариантные равенства малых степеней в программах, определенных над полем / С. М. Львов // Кибернетика. – 1988. - №1 – С. 108-110.

34. Letichevsky A. Discovery of invariant equalities in programs over data fields. / A. Letichevsky, M. Lvov. // Applicable Algebra in Engineering, Communication and Computing. – 1993. – №4. – С. 269–286.

35. Lvov M. S. About one algorithm of program polynomial invariants generation. / M. S. Lvov // Workshop on Invariant Generation, WING 2007 : Hagenberg, Austria: June 25–26. - 2007. – С. 85-99.

36. Львов М. С. Статический анализ физических размерностей переменных программ и его реализация в алгебраическом программировании / М. С. Львов. // Проблеми програмування. – 2015. - №2. – С. 3-12

37. Iterative methods of program analysis / A.Godlevskii, Y. Kapitonova, S. Krivoi, A. Letichevskii. // Cybernetics. – 1989. – №25(2). – С. 139–152.

38. Müller-Olm M., Seidl H. Computing polynomial program invariants / M. Müller-Olm, H. Seidl // Information Processing Letters. - 2004. - 91(5). – С. 233-244.

39. Львов М. С. Основи комп’ютерної алгебри та алгебраїчних обчислень : навч. посіб. / М. С. Львов. - Херсон : ТОВ «ВКФ «СТАР» ЛТД», 2018. - 238 с.

40. Летичевский А. А. Об одном подходе к анализу программ / А. А. Летичевский // *Кибернетика*. – 1979. - №6. – С. 1 – 8.

41. The experience of comparison of static security code analyzers / A. Markov, A. Fadin, V. Shvets, V. Tsirlov // International Journal of Advanced Studies. - 2015. - №5(3). – С. 55-63.

42. Multilevel metamodel for heuristic search of vulnerabilities in the software source code. / A. Markov, A. Fadin, V. Shvets, V. Tsirlov // International Journal of Control Theory and Applications. – 2016. - №9(30). – С. 313-320.

43. Аветисян А. Технологии статического и динамического анализа уязвимостей программного обеспечения / А. Аветисян, А. Белеванцев, И. Чукляев. // Вопросы кибербезопасности. – 2014. – №3. – С. 20–28.

44. Lopes R. Static Analysis tools, a practical approach for safety-critical software verification / R. Lopes, D. Vicente, N. Silva // ESA Special Publication. -2009 – 669 с.

45. Miller J. Design Development Test and Evaluation (DDT and E) Considerations for Safe and Reliable Human Rated Spacecraft Systems / J. Miller, J. Leggett, J. Kramer-White. - Volume II. – 2008. – 697 с.

46. ECSS-E-40 Part 2B: Software – Part 2: Document requirements definitions (DRDs) [Електронний ресурс]. – 2005. – Режим доступу до ресурсу: https://ecss.nl/standard/ecss-e-40-part-2b-software-part-2-document-requirements-definitions-drds/.

47. Lint [Електронний ресурс]. – Режим доступу до ресурсу: http://tack.sourceforge.net/olddocs/lint.pdf.

48. Emanuelsson P., Nilsson, U. A comparative study of industrial static analysis tools / P. Emanuelsson, U. Nilsson // Electronic notes in theoretical computer science. – 2008. - №217. – С. 5-21.

49. Checking system rules using system-specific, programmer-written compiler extensions / D. Engler, B. Chelf, A. Chou, S. Hallem // OSDI’00 Proceedings of the 4 th conference on Symposium on Operating System Design and Implementation. – 2000. - №4. – 16 с.

50. Why don't software developers use static analysis tools to find bugs? / B. Johnson, Y. Song, E. Murphy-Hill, R. Bowdidge // 35th International Conference on Software Engineering (ICSE). – 2013. – С. 672-681.

51. Franco J., Martin J. A History of Satisfiability / J. Franco, J. Martin// Handbook of satisfiability. – 2009. - №185, С. 3-74.

52. Cppcheck [Електронний ресурс]. – Режим доступу до ресурсу: http://cppcheck.sourceforge.net/

53. Clang Static Analyzer [Електронний ресурс]. – Режим доступу до ресурсу: https://clang-analyzer.llvm.org/

54. Clang-Tidy [Електронний ресурс]. – Режим доступу до ресурсу: https://clang.llvm.org/extra/clang-tidy/

55. Frama-C [Електронний ресурс]. – Режим доступу до ресурсу: http://frama-c.com/

56. Lint [Електронний ресурс]. – Режим доступу до ресурсу: http://tack.sourceforge.net/olddocs/lint.pdf - --- заменить

57. Parasoft C/C++test [Електронний ресурс]. – Режим доступу до ресурсу: https://www.parasoft.com/products/parasoft-c-ctest/c-c-static-analysis/

58. PC-Lint [Електронний ресурс]. – Режим доступу до ресурсу: https://gimpel.com/

59. Helix QAC [Електронний ресурс]. – Режим доступу до ресурсу: https://www.perforce.com/products/helix-qac

60. ReSharper [Електронний ресурс]. – Режим доступу до ресурсу: https://www.jetbrains.com/resharper/?gclid=EAIaIQobChMIoM-9gffs5wIVwQ8YCh0FOgw6EAAYASAAEgIzBPD_BwE

61. FxCop [Електронний ресурс]. – Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/visualstudio/code-quality/install-fxcop-analyzers?view=vs-2019

62. Roslyn Analyzers [Електронний ресурс]. – Режим доступу до ресурсу: https://docs.microsoft.com/en-US/visualstudio/code-quality/roslyn-analyzers-overview?view=vs-2019

63. Security Code Scan [Електронний ресурс]. – Режим доступу до ресурсу: https://security-code-scan.github.io/

64. Roslynator [Електронний ресурс]. – Режим доступу до ресурсу: https://marketplace.visualstudio.com/items?itemName=josefpihrt.Roslynator2019

65. CodeRush [Електронний ресурс]. – Режим доступу до ресурсу: https://www.devexpress.com/products/coderush/

66. Parasoft dotTEST [Електронний ресурс]. – Режим доступу до ресурсу: https://www.parasoft.com/products/parasoft-dottest/net-static-analysis/

67. . FindBugs [Електронний ресурс]. – Режим доступу до ресурсу: http://findbugs.sourceforge.net/

68. SpotBugs [Електронний ресурс]. – Режим доступу до ресурсу: https://spotbugs.github.io/

69. IntelliJ IDEA [Електронний ресурс]. – Режим доступу до ресурсу: https://www.jetbrains.com/idea/

70. SonarJava [Електронний ресурс]. – Режим доступу до ресурсу: https://www.sonarsource.com/java/

71. PyCharm [Електронний ресурс]. – Режим доступу до ресурсу: https://www.jetbrains.com/ru-ru/pycharm/

72. PyDev [Електронний ресурс]. – Режим доступу до ресурсу: https://www.pydev.org/

73. Pylint [Електронний ресурс]. – Режим доступу до ресурсу: https://pypi.org/project/pylint/

74. Coverity, "Coverity Static Analysis Data Sheet" (PDF). Synopsys.com. Retrieved 2019-07-15. [Електронний ресурс]. – Режим доступу до ресурсу: https://www.synopsys.com/content/dam/synopsys/sig-assets/datasheets/SAST-Coverity-datasheet.pdf

75. Klocwork Insight [Електронний ресурс]. – Режим доступу до ресурсу: https://marketplace.visualstudio.com/items?itemName=AlenZukich.KlocworkInsight

76. Checkmarx CxSuite [Електронний ресурс]. – Режим доступу до ресурсу: https://www.checkmarx.com/

77. DeepCode [Електронний ресурс]. – Режим доступу до ресурсу: https://www.researchgate.net/publication/229131551_DATALOG_SOLVE_A_Datalog-Based_Demand-Driven_Program_Analyzer

78. SapFix [Електронний ресурс]. – Режим доступу до ресурсу: https://engineering.fb.com/2018/09/13/developer-tools/finding-and-fixing-software-bugs-automatically-with-sapfix-and-sapienz/

79. Source{d} [Електронний ресурс]. – Режим доступу до ресурсу: https://github.com/src-d

80. CodeGuru [Електронний ресурс]. – Режим доступу до ресурсу: https://docs.aws.amazon.com/codeguru/latest/reviewer-ug/welcome.html

81. Infer [Електронний ресурс]. – Режим доступу до ресурсу: https://fbinfer.com/, https://engineering.fb.com/developer-tools/sapienz-intelligent-automated-software-testing-at-scale/

82. Sapienz [Електронний ресурс]. – Режим доступу до ресурсу: https://engineering.fb.com/developer-tools/finding-and-fixing-software-bugs-automatically-with-sapfix-and-sapienz/

83. Emanuelsson P., Nilsson U. A comparative study of industrial static analysis tools / P. Emanuelsson, U. Nilsson // Electronic notes in theoretical computer science. - 2008. – №217. – С. 5-21.

84. PolySpace Verifier, Polyspace Static Analysis [Електронний ресурс]. – Режим доступу до ресурсу: http://www.mathworks.com/products/polyspace/index.html?s_cid=psr_prod.

85. Static Code Analysis [Електронний ресурс]. – Режим доступу до ресурсу: http://www.coverity.com/html/prod_prevent.html.

86. KlocworkEmbeddedSystems [Електронний ресурс]. – Режим доступу до ресурсу: http://www.iplbath.com/pdf/klocwork/KlocworkEmbeddedSystems.pdf.

87. A light-weight static approach to analyzing UML behavioral properties / Yu. Lijun, R. France, I. Ray, K. Lano // 12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007). - 2007, July. – IEEE. – С. 56-63.

88. Counterexample-guided abstraction refinement for symbolic model checking / [E. Clarke, O, Grumberg, S. Jha, at all.]. // *Journal of the ACM (JACM)*/ - 2003. - №*50*(5). – P. 752-794.

89. Graf S., Saidi H. Construction of abstract state graphs with PVS / S. Graf, H. Saidi // International Conference on Computer Aided Verification. Springer, Berlin, Heidelberg. – 1997. - P. 72-83.

90. Ball T., Rajamani S. The SLAM project: Debugging system software via static analysis / T. Ball, S. Rajamani // Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages. – 2002. - pp. 1-3.

91. The software model checker b last / D. Beyer, T. Henzinger, R. Jhala, R. Majumdar // International Journal on Software Tools for Technology Transfer. - 2007. - №9(5) – pp. 505-525.

92. Software model checking via large-block encoding / [D. Beyer, A. Cimatti, A. Griggio, at all.] // Formal Methods in Computer-Aided Design. – 2009. - pp. 25-32.

93. McMillan K. L. Lazy abstraction with interpolants / K. L. McMillan // International Conference on Computer Aided Verification. Springer, Berlin, Heidelberg. - 2006. - pp. 123-136.

94. Lazy abstraction / T. A. Henzinger, R. Jhala, R. Majumdar, G. Sutre, // Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages. - 2002. -pp. 58-70.

95. Abstractions from proofs / Henzinger, T. A., Jhala, R., Majumdar, R., & McMillan, K. L. // ACM SIGPLAN Notices. - 2004. - №39(1). – pp. 232-244.

96. Steffen B. Data flow analysis as model checking / B. Steffen // International Symposium on Theoretical Aspects of Computer Software. Springer, Berlin, Heidelberg. – 1991. - pp. 346-364.

97. Gulwani S., Tiwari A. Combining abstract interpreters / S. Gulwani, A. Tiwari // ACM SIGPLAN Notices. - 2006. - №41(6). - pp. 376-386.

98. Lerner S. Composing dataflow analyses and transformations / S. Lerner, D. Grove, C. Chambers // ACM SIGPLAN Notices. - 2002. - №37(1). - pp. 270-282.

99. Fischer J. Joining dataflow with predicates / J. Fischer, R. Jhala, R. Majumdar, // ACM SIGSOFT Software Engineering Notes. – 2005. - №30(5). - pp. 227-236.

100. Beyer D. Program analysis with dynamic precision adjustment / D. Beyer, T. Henzinger, G. Théoduloz // 23rd IEEE/ACM International Conference on Automated Software Engineering. – 2008. - pp. 29-38

101. Kroening D. Decision Procedures. An Algorithmic Point of View / D. Kroening, O. Strichman. – Berlin: Springer-Verlag Berlin Heidelberg, 2008. – 306 p.

102. Efficient satisfiability modulo theories via delayed theory combination / [M. Bozzano, R. Bruttomesso, A. Cimatti, at all.] // International Conference on Computer Aided Verification. Springer, Berlin, Heidelberg. - 2005. - pp. 335-349.

103. Schmidt D. A. Data flow analysis is model checking of abstract interpretations / D. A. Schmidt // Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages. – 1998. - pp. 38-48.

104. Jones N. D., Muchnick S. S. A flexible approach to interprocedural data flow analysis and programs with recursive data structures / N. D. Jones, S. S. Muchnick // Proceedings of the 9th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. - 1982. - pp. 66-74.

105. M. Sagiv Parametric shape analysis via 3-valued logic / M. Sagiv, T. Reps, R. Wilhelm // ACM Transactions on Programming Languages and Systems (TOPLAS). – 2002. - №24(3). – pp. 217-298.

106. Beyer D. Lazy shape analysis / D. Beyer, T. A. Henzinger, G. Théoduloz // International Conference on Computer Aided Verification. Springer, Berlin, Heidelberg. - 2006. - pp. 532-546.

107. Wonisch D. Block abstraction memoization for CPAchecker / D. Wonisch / International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer, Berlin, Heidelberg. - 2012. - pp. 531-533.

108. Floyd R. W. Assigning Meanings to Programs / R. W. Floyd // Proceedings of Symposium on Applied Mathematics, Vol. 19, J.T. Schwartz (Ed.), American Mathematical Society, Providence. – 1967. - P. 19-32.

109. Hoare C. A. R. An axiomatic basis for computer programming / C. A. R. Hoare //Communications of the ACM. – 1969. – T. 12. – №.10. – C. 576-580.

110. Cocke J., Schwartz J.T. Programming Language and their Compilers / J. Cocke, J.T. Schwartz // Courant Institute of Mathem. Sciences. – New York University. – 1970. – N.Y. – P. 3–21.

111. Allen F.E. Interprocedural analysis / F. E. Allen // ACM SIGPLAN Notices. – 1976. – Vol. 6, N 7. – P. 23 – 31.

112. Летичевский А.А. Эквивалентность и оптимизация программ / А.А. Летичевский // Труды междунар. симпозиума по теоретическому программированию. – Новосибирск, 1972. – С. 166–180.

113. Kildall G.A. A unified approach to program optimization / G.A. Kildall // Conf. Rec. of ACM Symp. on Prince. of Program Languages, Boston, Massachusetts, Oktober 1–3, 1973. – P. 194–206.

114. Caplain M. Finding Invariant Assertions for Proving Programs / M. Caplain // Proc. of the intern. Conf. on Reliable Software, Los Angeles, California, April 21 – 23 1975. ACM: New York, NY, USA - 1975.- pp. 165-171.

115. Kam J.B., Ullman D.J. Monotone data flow analysis frame works / J.B. Kam, D.J. Ullman // Acta Inform. – 1978. – Vol. 3. – P. 305–318.

116. German S., Wegbreit B. A synthesizer of inductive assertions / S. German, B. Wegbreit // IEEE Transactions on Software Engineering. – 1975. - №1(1). – P. 68–75.

117. B. Wegbreit. The Synthesis of Loop Predicates / B. Wegbreit // Communications of the ACM. – 1974. - №17(2). - P. 102–112.

118. B. Wegbreit. Property extraction in well-founded property sets / B. Wegbreit // IEEE Transactions on Software Engineering. – 1975. - №1(3). – P. 270–285.

119. Research in Interactive ProgramProving Techniques. Technical report / B.Elspas, M. Green, K. Levitt, R. Waldinger. – Menlo Park, California, USA: Stanford Research Institute, 1972.

120. Katz S., Manna Z. Logical Analysis of Programs / S. Katz, Z. Manna // Communications of the ACM. – 1976. - №19(4). – P. 188–206.

121. Letichevsky A.A. About one approach to program analysis / A.A. Letichevsky // Cybernetics. - 1979. - № 6. - P. 1-8.

122. Iterative methods of program analysis / A.B. Godlevsky, Y.V. Kapitonova, S.L. Krivoy, A.A. Letichevsky // Cybernetics. – 1989. - №2.- P.9-19.

123. Кривой С. Л. О поиске инвариантных соотношений в программах / С. Л. Кривой // Математическая теория проектирования вычислительных машин. – Киев: Изд. Ин-та кибернетики АН УССР, 1978. – С. 35–51.

124. Sabelfeld V. K. Equivalente Transformationen f¨ur Flussdiagramme / V. K Sabelfeld // Acta Informatica. – 1978. – №10. – P. 127–155.

125. Кривой С. Л. Об одном алгоритме поиска инвариантных соотношений в программах / С. Л. Кривой // Кибернетика. – 1981. – № 5. – С. 12–18.

126. Lvov M. S. About one algorithm of program polynomial invariants generation / M. S. Lvov // Proc. Workshop on Invariant Generation: (Techn. rep.) Univ. of Linz / Eds. M. Giese, T. Jebelean. No 0707. – (RISC Report Series). Linz (Austria), 2007. – P. 85–99.

127. Львов М. С., Крекнін В. А. Нелінійні інваріанти лінійних циклів та власні поліноми лінійних операторів / М. С. Львов, В. А. Крекнін // Кибернетика и системный анализ. – 2012. – № 2. – С. 126–140.

128. Сабельфельд В.К. Учет свойств операций при глобальном анализе свойств программ / В.К. Сабельфельд // Математическая теория программирования. – Новосибирск: ВЦ СО АН СССР. – 1985. – С. 132–149.

129. Иткин В.Э. Логико–термальная эквивалентность схем программ / В.Э. Иткин // Кибернетика. – 1972. – № 1. – С. 5–27.

130. Cousot P., Halbwochs N. Discovery of Linear Restraints Among Variables of Program / P. Cousot, N. Halbwochs // Conf. Record of the 5–th Annual ACM Symposium on Princieples of Programming Languages. – USA. – 1978. – P. 84–96

131. Cousot P., Halbwochs N. Automatic Discovery of Linear Restraints among Variables of a Program. / P. Cousot, N. Halbwochs // Conference Record of the Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Tucson, Arizona, ACM Press, New York, NY. - 1978. – P. 84–97.

132. Karr M. Affine relationships among variables of a program / M. Karr // Acta Informatica. – 1976. - №6. – P. 133–151.

133. Cousot P., Cousot R. Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints / P. Cousot, R. Cousot // In Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Los Angeles, California. - ACM Press, New York, NY. - 1977. – P. 238–252.

134. Львов М.С. О вычислении инвариантов программ, интерпретированных над областью целостности / М.С. Львов // Кибернетика. – 1984. – №2. – С.23-28.

135. Львов М.С. Об инвариантных неравенствах для состояний схем программ, интерпретированных над векторным пространством / М.С. Львов // Кибернетика. – 1985. – №2.– С.111-112

136. . Львов М.С. Инвариантные неравенства в программах, интерпретированных над упорядоченными полями / М.С. Львов // Кибернетика. – 1986. – №5. – С.22-27

137. Львов М.С. Инвариантные равенства малых степеней в программах, определенных над полем/ М.С. Львов // Кибернетика. – 1988. –№1 . – С. 108-110

138. Letichevsky A., Lvov M. Discovery of Invariant Equalities in Programs over Data Fields / A. Letichevsky, M. Lvov // Applicable Algebra in Engineering, Communication and Computing. – 1993. – №4. – P. 21-29.

139. Müller-Olm M., Seidl H. Precise interprocedural analysis through linear algebra / M. Müller-Olm, H. Seidl // Proc. of Symposium on Principles of Programming Languages. - Venice, Italy, January 14-16, 2004. ACM: New York, NY, USA, 2004. - P. 330-341.

140. Müller-Olm M., Seidl H. Computing polynomial program invariants / M. Müller-Olm, H. Seidl // Inf. Process. Lett. Elsevier North-Holland, Inc. Amsterdam. - 2004.- № 91(5). - P. 233-244.

141. Kapur D. Automatically Generating Loop Invariants Using Quantifier Elimination–Preliminary Report / D. Kapur // IMACS Intl. Conf. on Applications of Computer Algebra. – 2004.

142. Sankaranarayanan S. Non-linear loop invariant generation using Gröbner bases / S. Sankaranarayanan, H. Sipma, Z. Manna // Proc. of Symposium on Principles of Programming Languages. - Venice, Italy, January 14-16, 2004. ACM: New York, NY, USA. - 2004.- P. 318-329.

143. Rodríguez-Carbonell E., Kapur D. Automatic generation of polynomial loop invariants: algebraic foundations / E. Rodríguez-Carbonell, D. Kapur // Proc. Of International Symposium on Symbolic and Algebraic Computation.- Santander, Spain, July 4-7, 2004.- ACM: New York, NY, USA 2004.- P. 266-273.

144. Rodríguez-Carbonell E., Kapur D. Automatic generation of polynomial invariants of bounded degree using abstract interpretation / E. Rodríguez-Carbonell, D. Kapur // Sci. Comput. Program..- Elsevier North-Holland, Inc. Amsterdam, The Netherlands.- 2007. - №64(1). - P.54-75.

145. Kovács L. I., Jebelean T. An Algorithm for Automated Generation of Invariants for Loops with Conditionals / L. I. Kovács, T. Jebelean // Proc. of Intern. Symposium on Symbolic and Numeric Algorithms for Scientific Computing . – Timisoara, Romania, 25-29 Sept. 2005. IEEE Computer Society: 2005.- P. 245-249.

146. Львов М.С. Полиномиальные инварианты линейных циклов / М.С.Львов // Кибернетика и системный анализ. –2010 - № 4. – С. 159–168.

147. Львов М.С., Крекнин В.А. Собственные полиномы линейных операторов и полиномиальные инварианты линейных циклов программ. / В.А. Крекнин, М.С.Львов // Науковий часопис національного педагогічного університету ім. М.П.Драгоманова. – 2010. – № 11. – С. 150–169.

148. Робинсон А. Введение в теорию моделей и метаматематику алгебры / .А. Робинсон. - М.:"Наука", ГЗФМЛ, 1967. – 376 с.

149. Кривой С.Л., Ракша С.Г. Поиск инвариантных линейных зависимостей в программах / С.Л. Кривой, С.Г. Ракша // Кибернетика. –1984. – №6. – С. 23–28.

150. Итеративные методы анализа программ. Равенства и неравенства / А.Б. Годлевский, Ю.В. Капитонова, С.Л. Кривой, А.А. Летичевский // Кибернетика. - 1990. - №3. - С. 1-10.

151. Львов М.С. Метод доказательства инвариантности линейных неравенств для линейных циклов / М.С. Львов // Кибернетика и системный анализ. – 2014. – № 4. – С. 80-85.

152. Годлевский А.Б., Кривой С.Л. Применение техники смешанных вычислений к построению эффективных алгоритмов унификации и приведения выражений / А.Б. Годлевский, С.Л. Кривой // Тезисы докл. Всесоюз. конф. "Методы трансляции и конструирования программ". – Новосибирск, 1988. – С. 59–62.

153. Hoare T. The verifying compiler: A Grand challenge for computing research / T. Hoare // J. of the ACM. – 2003. – № 50(1) – P. 63–69.

154. Invariant semidefinite programs / C. Bachoc, D. C. Gijswijt, A. Schrijver, F. Vallentin // Handbook on semidefinite, conic and polynomial optimization. – Springer US, 2012. – P. 219-269.

155. Müller-Olm M. Interprocedurally analyzing polynomial identities / M. Müller-Olm, M. Petter, H. Seidl // STACS 2006. – Springer Berlin Heidelberg, 2006. – С. 50-67.

156. Rodríguez-Carbonell E., Kapur D. Generating all polynomial invariants in simple loops / E. Rodríguez-Carbonell, D. Kapur //Journal of Symbolic Computation. – 2007. – №. 42(4). – P. 443-476.

157. Sipma H. B., Manna Z. Non-linear loop invariant generation using Gröbner bases / H. B. Sipma, Z. Manna //ACM SIGPLAN Notices. – 2004. – №. 39(1). – P. 318-329.

158. Furia C. A.. A survey of loop invariants / Furia C. A., Meyer B., Velder S. // CoRR. – 2012.

159. Ван-дер-Варден Б. Алгебра. Изд. 2-ое.` / Б. Ван-дер-Варден. – М.: ГРФМЛ, 1979. – 624 с.

160. Ходж В., Пидо Д. Методы алгебраической геометрии. Том 1. / В. Ходж, Д. Пидо. - М.: ИЛ, 1954. - 462 с.

161. Геометрическая теория инвариантов. / Дьедонне Ж., Керрол Дж, Мамфорд Д. - М.:Мир, 1974.- 280 с.

162. Львов М.С. О структуре полиномиальных инвариантов линейных циклов / М.С. Львов // Кибернетика и системный анализ. — 2015. — № 51(3) - С. 143-156.

163. Компьютерная алгебра: символьные и алгебраические вычисления / Под ред. Б.Бухбергера, Дж. Коллинза, Р.Лооса. – М.:Мир, 1986. - 392 с.

164. Lvov M. The Static Analysis of Linear Loops [Електронний ресурс] / M. Lvov, Y. Tarasich // CEUR Workshop Proceedings. – 2015. – Режим доступу до ресурсу: http://ceur-ws.org/Vol-1356/paper_79.pdf.

165. ACL2 Version 8.3 [Електронний ресурс]. – Режим доступу до ресурсу: http://www.cs.utexas.edu/~moore/acl2/.

166. Kaufmann M., Moore J. S. Design Goals for ACL2. / M. Kaufmann, J. S. Moore. // Proc. of 3-rd International School and Symposium on Formal Techniques in Real Time and Fault Tolerant Systems. - 1994. - P. 92-117.

167. E-SETHEO [Електронний ресурс]. – Режим доступу до ресурсу: http://www4.informatik.tu-muenchen.de/~schulz/WORK/e-setheo.html.

168. The E Theorem Prover [Електронний ресурс]. – Режим доступу до ресурсу: http://www.eprover.org/.

169. Bachmair L., Ganzinger H. Rewrite-Based Equational Theorem Proving with Selection and Simplification. / L. Bachmair, H. Ganzinger. // Journal of Logic and Computation. - 1994. - №4(3). – P. 217 - 247.

170. The KeY to Software Correctness [Електронний ресурс]. – Режим доступу до ресурсу: http://www.key-project.org/.

171. Beckert B. Verification of Object-Oriented Software. The KeY Approach / B. Beckert, R. Hähnle, P. Schmitt., 2007. – 658 с. – (4334).

172. Riazanov A., Voronkov A. The Design and Implementation of Vampire. / A. Riazanov, A. Voronkov // AI Communications. – 2002. - №15(2). – P. 91 - 110.

173. Waldmeister [Електронний ресурс]. – Режим доступу до ресурсу: http://www.mpi-inf.mpg.de/~hillen/waldmeister/.

174. Darwin [Електронний ресурс]. – Режим доступу до ресурсу: ttp://combination.cs.uiowa.edu/Darwin/.

175. PVS [Електронний ресурс]. – Режим доступу до ресурсу: http://pvs.csl.sri.com/

176. HOL [Електронний ресурс]. – Режим доступу до ресурсу: https://hol-theorem-prover.org/

177. Isabelle [Електронний ресурс]. – Режим доступу до ресурсу: https://isabelle.in.tum.de/

178. Coq [Електронний ресурс]. – Режим доступу до ресурсу: https://coq.inria.fr/

179. CVC4 [Електронний ресурс]. – Режим доступу до ресурсу: https://cvc4.github.io/

180. MathSAT5 [Електронний ресурс]. – Режим доступу до ресурсу: https://mathsat.fbk.eu/

181. QEPCAD [Електронний ресурс]. – Режим доступу до ресурсу: https://www.usna.edu/CS/qepcadweb/B/QEPCAD.html

182. Singular W. Decker, G.-M. Greuel, G. Pfister and H. Sch¨onemann. Singular 4- 0-2 — A computer algebra system for polynomial computations. [Електронний ресурс]. – Режим доступу до ресурсу: http://www.singular.uni-kl.de

183. COCOA [Електронний ресурс]. – Режим доступу до ресурсу: http://cocoa.dima.unige.it/

184. CoCoALib [Електронний ресурс]. – Режим доступу до ресурсу: http://cocoa.dima.unige.it/cocoalib/

185. FlatZinc [Електронний ресурс]. – Режим доступу до ресурсу: https://www.gecode.org/flatzinc.html

186. STP [Електронний ресурс]. – Режим доступу до ресурсу: https://stp.github.io/

187. RedLog [Електронний ресурс]. – Режим доступу до ресурсу: https://www.redlog.eu/

188. Satallax [Електронний ресурс]. – Режим доступу до ресурсу: https://www.ps.uni-saarland.de/~cebrown/satallax/

189. Gernot S., Andreas W. E-SETHEO: Design, configuration and use of a parallel automated theorem prover / S. Gernot; W. Andreas. // Australasian Joint Conference on Artificial Intelligence. Springer, Berlin, Heidelberg. - 1999. - P. 231-243.

190. Niklas S., Niklas E. Minisat v1. 13-a sat solver with conflict-clause minimization. / S. Niklas, E. Niklas // SAT. – 2005. - №53. – P. 1 - 2.

191. Jürgen C. SMTInterpol: An interpolating SMT solver / C. Jürgen, H. Jochen, N. Alexander // International SPIN Workshop on Model Checking of Software. Springer, Berlin, Heidelberg. - 2012. - P. 248-254.

192. TPS: A hybrid automatic-interactive system for developing proofs / A. B. Peter, B. E. Chad // Journal of Applied Logic. – 2006. - №4(4). – P. 367-395.

193. Paradox 3.0., https://web.archive.org/web/20070108005818/ http://www.cs.chalmers.se/~koen/folkung/

194. Tammet T. Gandalf version c-1.0c Reference Manual / T. Tammet., 1997.

195. Vampire [Електронний ресурс]. – Режим доступу до ресурсу: http://www.vprover.org/

196. Kovács L., Voronkov A. First-order theorem proving and Vampire. / L. Kovács, A. Voronkov // International Conference on Computer Aided Verification. Springer, Berlin, Heidelberg. - 2013. - P. 1-35.

197. Fourier–Motzkin elimination [Електронний ресурс]. – Режим доступу до ресурсу: http://en.wikipedia.org/wiki/Fourier%E2%80%93Motzkin_elimination

198. Collins G. E. Quantifier elimination by cylindrical algebraic decomposition — twenty years of progress / G. E. Collins // Quantifier elimination and cylindrical algebraic decomposition (Ed. B. F. Caviness and J. R. Johnson). — New York: Springer–Verlag, 1998. — P. 8–23.

199. Omega Test [Електронний ресурс]. – Режим доступу до ресурсу: http://www.umiacs.umd.edu/~hismail/Omega/node3.html

200. Badros G. The Cassowary linear arithmetic constraint solving algorithm / G. Badros, A. Borning, P. Stuckey // ACM transactions on computer-human interaction (TOCHI). — 2001. — №8(4) — P. 267–306.

201. Lvov M. S., Peschanenko V. S. Trapezoid method for solving systems of linear inequalities and its implementation in insertion modeling / M. S.Lvov, V. S. Peschanenko //Cybernetics and Systems Analysis. – 2012. – Т. 48. – №. 6. – С. 931-942

202. Lvov M. S. Algebraic approach to the problem of solving systems of linear inequalities / M. S.Lvov // Cybernetics and Systems Analysis. – 2010. – Т. 46. – №. 2. – С. 326-338.

203. IMS [Електронний ресурс]. – Режим доступу до ресурсу: http://www.apsystems.org.ua/node/7

204. Letichevsky A., Letichevsky A. (jr), Peschanenko V. APS and Tools / A. Letichevsky, A. Letichevsky (jr), V. Peschanenko // Herald of the V.N.Karazin Kherkiv Nitional University. – 2010. - p. 165-173.

205. Letichevsky A. Algebra of behavior transformations and its applications / A. Letichevsky // Structural theory of Automata, Semigroups, and Universal Algebra, NATO Science Series II. Mathematics, Physics and Chemistry. – 2005. – № 207. – P. 241–272.

206. Львов М.С. Синтез інтерпретаторів алгебраїчних операцій в розширеннях багатосортних алгебр / М.С.Львов // Вісник Харьківського національного университету. - 2009.-№847.-С.221-238. - (Серія «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління»).

207. Львов М.С. Метод спадкування при реалізації алгебраїчних обчислень в математичних системах навчального призначення / М.С.Львов// Системи управління, навігації та зв'язку. - К: ЦНДІ НіУ, 2009.- Вип.3 (11).- С.120-130.

208. Песчаненко В.С. Об одном подходе к проєктированию алгебраических типов данных / В.С. Песчаненко // Проблеми програмування. – 2006. – №2 – 3. – С. 626 – 634.

209. . Buchberger B. Basic features and development of the Critical-pair/Completion procedure / B. Buchberger // Lect. Notes Comput. Sci. — 1985. — Vol. 202. — P. 1—45.

210. Бухбергер Б., Лоос Р. Упрощение алгебраических выражений / Б. Бухбергер, Р. Лоос // Компьютерная алгебра. Символьные и алгебраические вычисления. — М.: Мир. 1986. — С. 23—65.

211. Knuth D.E., Bendix P.B. Simple Word Problems in Universal Algebra / D.E. Knuth, P.B. Bendix // Automation of Reasoning. – 1983. – P. - 342–376.

212. Book R. V., Otto F. String-rewriting systems. / R. V. Book, F. Otto // String-Rewriting Systems. Springer, New York, NY. - 1993. - p. 35-64.

213. Реалізація алгоритму побудови КФ ЛНПФ [Електронний ресурс]. – Режим доступу до ресурсу: https://drive.google.com/drive/folders/1jaK0S5rTI75ZX5qUEUNbQ8I3LdT1W9JY?usp=sharing

214. Park D. Concurrency and automata on infinite sequences / D. Park // Theoretical computer science. Springer, Berlin, Heidelberg. - 1981. - P. 167-183.

215. The canonical forms of logical formulas off the data types / M. Lvov, V. Peschanenko, O. Letychevskyi, Yu. Tarasich // 13th International Conference Theoretical and Applied Aspect of Program Systems Development. – 2016.

216. The canonical forms of logical formulae over the data types and their using in programs verification / Lvov, M., Peschanenko, V., Letychevskyi, O., Tarasich, Y. // CEUR Workshop Proceedings. – 2017. - №1844. - P. 536–554.

217. Algorithm and Tools for Constructing Canonical Forms of Linear Semi-Algebraic Formulas / [M. Lvov, V. Peschanenko, Y. Tarasich та ін.]. // Cybernetics and Systems Analysis. – 2018. – №54. – С. 993–1002.

218. Formalization and algebraic modeling of tokenomics projects [Електронний ресурс] / [O. Letychevskyi, V. Peschanenko, V. Radchenko та ін.] // CEUR Workshop Proceedings. – 2019. – Режим доступу до ресурсу: http://ceur-ws.org/Vol-2393/paper_409.pdf.

219. Our Approach to Formal Verification of Token Economy Models / O.Letychevskyi, V. Peschanenko, M. Poltoratskyi, Y. Tarasich, 2020. – (Communications in Computer and Information Science). – (1175 CCIS). – С. 348 – 363.

220. Platform for Modeling of Algebraic Behavior: Experience and Conclusions / Letychevskyi, O., Peschanenko, V., Poltoratskyi, M., Tarasich, Y...// CEUR Workshop Proceedings. - 2020. – Режим доступу до ресурсу: http://ceur-ws.org/Vol-2732/20200042.pdf

ДОДАТКИ

Додаток 1

Список публікацій здобувача

Основні наукові результати дисертації

1. Lvov M. The Static Analysis of Linear Loops [Електронний ресурс] / M. Lvov, Y. Tarasich // CEUR Workshop Proceedings. – 2015. – Режим доступу до ресурсу: http://ceur-ws.org/Vol-1356/paper_79.pdf.

Особистий внесок здобувачки полягає в аналізі та описі основних результатів проблеми доказу та генерації поліноміальних інваріантних рівностей, визначенні нових засобів та алгоритмів статичного аналізу програм, а саме - нового алгоритму доказу інваріантних нерівностей.

2. The canonical forms of logical formulas off the data types / M. Lvov, V. Peschanenko, O. Letychevskyi, Yu. Tarasich // 13th International Conference Theoretical and Applied Aspect of Program Systems Development. – 2016. (тези)

Особистий внесок здобувачки полягає у представлені нових алгоритмів обчислення канонічної форми логічних формул над перелічуваним та множинним типами даних.

3. The canonical forms of logical formulae over the data types and their using in programs verification [Електронний ресурс] / M.Lvov, V. Peschanenko, O. Letychevskyi, Y. Tarasich // CEUR Workshop Proceedings. – 2017. – Режим доступу до ресурсу: http://ceur-ws.org/Vol-1844/10000536.pdf.

Особистий внесок здобувачки полягає у стислому огляді засобів спрощення формул, представлені нового алгоритму обчислення канонічної форми логічних формул над упорядкованими типами даних.

4. Algorithm and Tools for Constructing Canonical Forms of Linear Semi-Algebraic Formulas / [M. Lvov, V. Peschanenko, Y. Tarasich and all.]. // Cybernetics and Systems Analysis. – 2018. – №54. – С. 993–1002.

Особистий внесок здобувачки полягає в тестуванні засобів спрощення формул та аналізі їх основних функціональних особливостей; представленні нового алгоритму побудови канонічних форм лінійних напівалгебраїчних формул.

Апробація матеріалів дисертації

5. Formalization and algebraic modeling of tokenomics projects [Електронний ресурс] / [O. Letychevskyi, V. Peschanenko, V. Radchenko та ін.] // CEUR Workshop Proceedings. – 2019. – Режим доступу до ресурсу: http://ceur-ws.org/Vol-2393/paper_409.pdf.

Особистий внесок здобувачки полягає у застосуванні розроблених алгоритмів для моделювання визначених задач.

6. Our Approach to Formal Verification of Token Economy Models / O.Letychevskyi, V. Peschanenko, M. Poltoratskyi, Y. Tarasich, 2020. – (Communications in Computer and Information Science). – (1175 CCIS). – С. 348 – 363.

Особистий внесок здобувачки полягає у застосуванні розроблених алгоритмів для моделювання визначених задач.

7. Platform for Modeling of Algebraic Behavior: Experience and Conclusions / Letychevskyi, O., Peschanenko, V., Poltoratskyi, M., Tarasich, Y...// CEUR Workshop Proceedings. - 2020. – Режим доступу до ресурсу: http://ceur-ws.org/Vol-2732/20200042.pdf

Особистий внесок здобувачки полягає у застосуванні розроблених алгоритмів для моделювання визначених задач.

Апробація матеріалів дослідження на конференціях, семінарах, круглих столах тощо

Рівень заходу й його назва	Місце й дата проведення	Форма участі
Міжнародна наукова конференція ICTERI 2015	м. Львів, травень 2015 р.	Очна, публікація у матеріалах конференції
Міжнародна наукова конференція «Теоретичні та прикладні аспекти побудови програмних систем»	м. Київ, 5-9 грудня 2016 р.	Заочна, публікація у матеріалах конференції
Міжнародна наукова конференція ICTERI 2017	м. Київ, 15-19 травня 2017 р.	Очна, публікація у науковому виданні, що індексується Scopus
Міжнародна наукова конференція сучасна інформатика: проблеми, досягнення та перспективи розвитку	м. Київ, 13-15 грудня, 2017 р.	Очна, Доповідь на конференції
Міжнародна наукова конференція ICTERI 2019	м. Херсон, травень, 2019 р.	Очна, 1 публікація у науковому виданні, що індексується Scopus
Міжнародна наукова конференція ICTERI 2020	м. Харків, жовтень, 2020 р.	Очна, 1 публікація у науковому виданні, що індексується Scopus

Додаток 2

formula 1----------

```
Exist x3 ((1 = -1 * y + x)&(0 <= R(x3)&
R(x3) <= 3|/9 <= R(x3)&
R(x3) <= 18)&(11 <= F(x3)&
F(x3) <= 20|/2 <= F(x3)&
F(x3) <= 5)&-1 * x3 + y <= (-1))
```

formula 2----------

```
Exist x3 ((1 = -1 * y + x)&(0 <= R(x3)&
R(x3) <= 3|/9 <= R(x3)&
R(x3) <= 18)&(11 <= F(x3)&
F(x3) <= 20|/2 <= F(x3)&
F(x3) <= 5)&-1 * x3 + y <= (-1)&
-1 * x3 + y <= (-2)|/(1 = -1 * y + x)&
(-1 = -1 * x3 + y)&(7 <= R(x3)&
R(x3) <= 16|/-2 <= R(x3)&
R(x3) <= 1)&(13 <= F(x3)&
F(x3) <= 22|/4 <= F(x3)&
F(x3) <= 7))
```

formula 3----------

```
Exist x3 ((1 = -1 * y + x)&(0 <= R(x3)&
R(x3) <= 3|/9 <= R(x3)&
R(x3) <= 18)&(11 <= F(x3)&
F(x3) <= 20|/2 <= F(x3)&
F(x3) <= 5)&-1 * x3 + y <= (-1)&
-1 * x3 + y <= (-2)|/(1 = -1 * y + x)&
(-1 = -1 * x3 + y)&(5 <= R(x3)&
R(x3) <= 14|/-4 <= R(x3)&
R(x3) <= (-1))&(6 <= F(x3)&
F(x3) <= 9|/15 <= F(x3)&
F(x3) <= 24))
```

formula 4----------

```
Exist x3 ((1 = -1 * y + x)&
(0 <= R(x3)&R(x3) <= 3|/
9 <= R(x3)&R(x3) <= 18)&
(11 <= F(x3)&F(x3) <= 20|/
2 <= F(x3)&F(x3) <= 5)&
-1 * x3 + y <= (-1)&-1 * x3 + y <= (-2)|/
(1 = -1 * y + x)&(-1 = -1 * x3 + y)&
(1 <= R(x3)&R(x3) <= 10|/
-8 <= R(x3)&R(x3) <= (-5))&
(10 <= F(x3)&F(x3) <= 13|/
19 <= F(x3)&F(x3) <= 28)
)
```

formula 5----------

```
Exist (x0,x2,x4)(
F(x) <= 4&1 <= F(x)&
(F(x4) = 4)&(F(x2) = 4)&
(F(x0) = 4))
```

formula 6----------

```
Exist x6(F(x) <= 5&
2 <= F(x)&(-1 * x + x6 <= (-1)|/
-1 * x6 + x <= (-1))&(F(x6) = 4))
```

formula 7----------

```
Exist (x!11 : int) (Forall (x!9 : int) (
(~(x!11 + -1 * x!9 <= (-1))&~(-1 * x!11 + x!9
<= (-1))|/
(Mesi(x!11) = E)&(Mesi(x!9) = I))&
(~(x!11 + -1 * x!9 <= 0)|/~(x!9 + -1 * x!11 <=
0)|/
(Mesi(x!11) = E))))
```

formula 8----------

```
~(x + -1 * y <= (-4))&~(y + -1 * x <= (-4))&
~(-1 * y + -1 * x <= (-4))&~(x + y <= (-4))&
~(x + -1 * y <= (-2))&~(y + -1 * x <= (-2))&
~(-1 * y + -1 * x <= (-2))&~(x + y <= (-2))|/
~(x + -1 * y <= 8)&~(y + -1 * x <= (-16))&
~(-1 * y + -1 * x <= (-16))&~(x + y <= 8)&
~(x + -1 * y <= (-2))&~(y + -1 * x <= (-2))&
~(-1 * y + -1 * x <= (-2))&~(x + y <= (-2))
```

formula 9----------

```
~(x + -1 * y <= (-4))&~(y + -1 * x <= (-4))&
~(-1 * y + -1 * x <= (-4))&~(x + y <= (-4))&
~(x + -1 * y <= 0)&~(y + -1 * x <= (-8))&
~(-1 * y + -1 * x <= (-8))&~(x + y <= 0)|/
~(x + -1 * y <= 8)&~(y + -1 * x <= (-16))&
~(-1 * y + -1 * x <= (-16))&~(x + y <= 8)&
~(x + -1 * y <= 0)&~(y + -1 * x <= (-8))&
~(-1 * y + -1 * x <= (-8))&~(x + y <= 0)
```

formula 10----------
-

```
Exist v ((v=1)&(x=v))
```

Додаток 3

Реалізація алгоритму побудови канонічних форм лінійних напівалгебраїчних формул. Основна функція build_concrete_values (для довільної формули (з функціоналами і т.д.))

```
     node_ptr build_concrete_values(fpl_ptr &fpl,node_ptr
t,node_ptr &order){
      if (t->get_mark()==fpl->intmrk)
            return t;
      rebuild_formula(fpl,t,false);
      node_ptr make_from_structure(fpl_ptr &fpl,node_ptr &t);
      t=make_from_structure(fpl,t);
      name_ptr ac_list = fpl->find_name(char_ptr("ac_list"));
      node_ptr old_val = ac_list->val();
      ac_list->val()=fpl->make_hesh_formula(111,"ARRAY(()+ (),
() $ (), 0,  nil,  nil),(()* (), ()^ (), 1,   0,  nil),(()& (),
~(()), 1,   0,   0),(()|/(), ~(()), 0,   1,   1))",0);
     #ifdef DEBUG
      std::cout<<"bcv1:"<<fpl->printNodeInBuf(t)<<std::endl;
     #endif //DEBUG
      node_ptr p;
      eliminate_all_functionals(fpl,t,p);
     #ifdef DEBUG
      std::cout<<"bcv2:"<<fpl->printNodeInBuf(t)<<":"<<fpl-
>printNodeInBuf(p)<<std::endl;
     #endif //DEBUG
      node_ptr vv = leave_right_parts_of_ass(fpl,p);
      if (!vv.IsEmpty())
            t = fpl->make_exist(vv,t);
     #ifdef DEBUG
      std::cout<<"bcv2:"<<fpl->printNodeInBuf(t)<<std::endl;
     #endif //DEBUG
     #ifdef DEBUG
      std::cout<<"bcv2:"<<fpl->printNodeInBuf(t)<<std::endl;
     #endif //DEBUG
      collect_quantifiers(fpl,t);
      if (is_exist_term(fpl,t))
            t=fpl->vali(fpl->vali(t,1),1);
     #ifdef DEBUG
      std::cout<<"bcv2:"<<fpl->printNodeInBuf(t)<<std::endl;
     #endif //DEBUG
```

```
      eliminate_all_not(fpl,t);
     #ifdef DEBUG
      std::cout<<"bcv2:"<<fpl->printNodeInBuf(t)<<":"<<fpl-
>printNodeInBuf(p)<<std::endl;
     #endif //DEBUG
      t = dnf(fpl,t);
     #ifdef DEBUG
     std::cout<<"bcv3:"<<fpl->printNodeInBuf(t)<<std::endl;
     #endif
      eliminate_all_not(fpl,t);
     #ifdef DEBUG
     std::cout<<"bcv3:"<<fpl->printNodeInBuf(t)<<std::endl;
     #endif
      t= DisjCan(fpl,t);
     #ifdef DEBUG
      std::cout<<"bcv3:"<<fpl->printNodeInBuf(t)<<std::endl;
     #endif //DEBUG
      if (!p.IsEmpty())
            t = dnf(fpl,t);
      if (!fpl->is_or(t))
            t = fpl->make_or(t,fpl->make_nil());
      if (fpl->is_one(t)||fpl->is_zero(t)){
            ac_list->val() = old_val;
            return t;
      }
     #ifdef DEBUG
      std::cout<<"bcv4:"<<fpl->printNodeInBuf(t)<<std::endl;
     #endif //DEBUG
      node_ptr res = fpl->zero;
      while(fpl->is_or(t)){
            CFPLTYPE fp;
            if (!order.IsEmpty())
                  fp = get_type(fpl,order);
            node_ptr torder;
            if (fp.m_type==FPLTYPE_INT||fp.m_type==FPLTYPE_REAL)
                  torder = order;
            res = mrg(fpl,fpl-
>make_or(get_concrete_value(fpl,fpl->vali(t,0),torder),res));
     #ifdef DEBUG
      std::cout<<"bcv5:"<<fpl->printNodeInBuf(res)<<std::endl;
     #endif //DEBUG
            t = fpl->vali(t,1);
            if (!fpl->is_nil(t)&&!fpl->is_or(t))
                  t = fpl->make_or(t,fpl->make_nil());
```

```
  }
  if (!p.IsEmpty())
        res = subs(fpl,inverse_subs(fpl,p),res);
#ifdef DEBUG
  std::cout<<"bcv6:"<<fpl->printNodeInBuf(res )<<std::endl;
#endif //DEBUG
  eliminate_pminf(fpl,res);
#ifdef DEBUG
  std::cout<<"bcv7:"<<fpl->printNodeInBuf(res )<<std::endl;
#endif //DEBUG
  ac_list->val() = old_val;
  return res;
}
```

Додаток 4

Реалізація алгоритму побудови канонічних форм лінійних напівалгебраїчних формул. Функція побудови канонічної форми makeCanForm

```
node_ptr makeCanForm(fpl_ptr &fpl,node_ptr &t,node_ptr &order){
      if (fpl->is_or(t)){
            t = fpl->make_add(makeCanForm(fpl,fpl->vali(t,0),order),makeCanForm(fpl,fpl->vali(t,1),order));
      node_ptr try2mrg_intervals(fpl_ptr &fpl,node_ptr &t1,node_ptr &t2,bool&);
            bool blval=false;
            t = try2mrg_intervals(fpl,fpl->vali(t,0),fpl->vali(t,1),blval);
            return t;   }
      node_ptr f = fpl->one;
     #ifdef DEBUG
     cout<<"mfmtf1:"<<fpl->printNodeInBuf(t)<<endl;
     #endif
      collect_variables(fpl,t,f);
     #ifdef DEBUG
     cout<<"mfmtf1:"<<fpl->printNodeInBuf(f)<<endl;
     #endif
      if (fpl->is_one(f))
            return t;
      f = translate_seq(fpl,f,order);
      get_initial_state(fpl,f);
      make_attrs_mlt(fpl,t);
      return add_rec_eqs(fpl,t,f); }
```