УДК 519.683.2
DOI https://doi.org/10.32689/maup.it.2023.1.5

Василь КОСТИРКО
кандидат фізико–математичних наук, доцент кафедри комп'ютерних наук, Львівський торговельно–економічний університет, вул. Туган–Барановського, 10, Львів, Україна, індекс 79005 (vkostyrko@lute.lviv.ua)
ORCID: 0000-0002-6366-8695

Анатолій КОСТЕНКО
кандидат фізико–математичних наук, доцент, завідувач кафедри комп'ютерних наук, Львівський торговельно–економічний університет, вул. Туган–Барановського, 10, Львів, Україна, індекс 79005 (avak54@lute.lviv.ua)
ORCID: 0000-0002-2162-7852

Михайло ПЛЕША
кандидат фізико–математичних наук, доцент кафедри комп'ютерних наук, Львівський торговельно–економічний університет, вул. Туган–Барановського, 10, Львів, Україна, індекс 79005 (milan@lute.lviv.ua)
ORCID: 0009-0002-6496-1102

Vasyl KOSTYRKO
Candidate of Physical and Mathematical Sciences, Associate Professor at the Department of Computer Scinces, Lviv University of Trade and Economics, 10 Tuhan–Baranovskoho str., Lviv, Ukraine, postal code 79005 (vkostyrko@lute.lviv.ua)

Anatolij KOSTENKO
Candidate of Physical and Mathematical Sciences, Associate Professor, Head of the Department of Computer Sciences, Lviv University of Trade and Economics, 10 Tuhan–Baranovskoho str., Lviv, Ukraine, postal code 79005 (avak54@lute.lviv.ua)

Mykhaylo PLESHA
Candidate of Physical and Mathematical Sciences, Associate Professor at the Department of Computer Scinces, Lviv University of Trade and Economics, 10 Tuhan–Baranovskoho str., Lviv, Ukraine (milan@lute.lviv.ua)

Бібліографічний опис статті: Костирко, В., Костенко, А, Плеша, М. (2023). Застосування солверів z3 в системі верифікації Python–програм. *Інформаційні технології та суспільство*. Вип. 1 (7), 36–43. DOI: https://doi.org/10.32689/maup.it.2023.1.5

Bibliographic description of the article: Kostyrko, V., Kostenko, A., Plesha, M. (2023). Navchannia prohramuvanniu z zastosuvanniam structurovanykh blok–skhem ta shabloniv proektuvannia [Application of z3 solvers in the Python program verification system]. *Informatsiini tekhnolohii ta suspilstvo – Information technology and society,* 1 (7). 36–43. DOI: https://doi.org/10.32689/maup.it.2023.1.5

ЗАСТОСУВАННЯ СОЛВЕРІВ Z3 В СИСТЕМІ ВЕРИФІКАЦІЇ PYTHON–ПРОГРАМ

У роботі описана остання версія системи верифікації програм VerPro на мові Python. Для перевірки тотожної істинності умов верифікації вона доповнена застосуванням засобів популярної системи Z3 доведення теорем. ***Метою статті*** *є опис нових можливостей системи VerPro, які їй надали солвери Z3.* ***Наукова новизна.*** *Система VerPro є сучасною системою верифікації анотованих Python–програм, яка генерує умови коректності методом символьного виконання. VerPro надає користувачу сучасний діалоговий графічний віконний інтерфейс, побудований з використанням бібліотеки PyQt5. Спрощення цих умов базується на еквівалентних перетвореннях, які реалізує бібліотека ExprLib. ExprLib – це наша власна бібліотека, яка побудована на мові Python і не має інших залежностей. Бібліотека перетворює вирази у внутрішнє представлення, побудоване на базі класів мови Python. Реалізовані в системі VerPro еквівалентні перетворення часто можуть самі встановити тотожну істинність умов верифікації. Якщо ні, тоді такі умови VerPro передає системі доведення теорем Z3, яка або встановлює їх тотожню істинність, або будує контрприклад для гіпотези про тотожню істинність такої умови. Наявність контрприкладу свідчить про те, що анотація невірно описує функціональні властивості програми. Аналізуючи знайдений системою контрприклад, користувач може знайти причину некоректності і виправити програму і/або її анотацію. Система Z3 підключається до VerPro через програмний інтерфейс Z3ру.* ***Висновок.*** *Ідея побудови контрприкладів в системах верифікації програм реалізована вперше. Система доведення теорем Z3 раніше в таких системах не застосовувалася. Система VerPro надає приклад вдалого використання сучасної системи доведення теорем не лише для встановлення коректності анотованої програми, але й для пошуку рішення при невдалій верифікації. Дальший розвиток системи передбачає типізацію значень символьних параметрів та побудову інваріантів.*

Ключові слова: *верифікація програми; умова коректності; символьне виконання; контрприклад; солвер Z3.*

APPLICATION OF Z3 SOLVERS IN THE VERIFICATION SYSTEM FOR PYTHON PROGRAMS

The work describes the latest version of the VerPro program verification system in Python. In order to check the identical truth of the verification conditions, it was supplemented by the solvers of the popular Z3 theorem proving system. ***The purpose of the article*** *is to describe the new capabilities of the VerPro system, provided by the Z3 solvers.* ***Scientific novelty.*** *The VerPro system is a modern verification system for annotated Python programs that generates correctness conditions by the method of symbolic execution. VerPro provides the user with a modern dialog graphical window interface built using the PyQt5 library. The simplification of these conditions is based on equivalent transformations implemented by the ExprLib library. ExprLib is our own library built in Python without no other dependencies. The library transforms expressions into an internal representation built on the Python language classes. Equivalent transformations implemented in the VerPro system can often establish the identical truth of the verification conditions themselves. If not, then VerPro passes such conditions to the Z3 theorem proving system, which either establishes their identical truth, or builds a counterexample for the hypothesis of the identical truth of such a condition. The presence of a counterexample indicates that the annotation incorrectly describes the functional properties of the program. Analyzing the counterexample found by the system, the user can find the cause of the incorrectness and change the program and/or its annotation. The Z3 system connects to the VerPro via the Z3py software interface.* ***Conclusion.*** *The idea of building counterexamples in program verification systems was implemented for the first time. The Z3 theorem proving system was not previously used in such systems. The VerPro system provides an example of a successful use of a modern theorem proving system not only to establish the correctness of an annotated program, but also to find a solution in case of failed verification. Further development of the system involves the assignment of types for symbolic parameter values and the construction of invariants.*

Key words: *program verification, correctness condition; symbol execution; counterexample; Z3 solver.*

Актуальність проблеми

Система *VerPro* призначена для верифікації програм на невеликій підмножині мови Python [K1]. Ця підмножина включає як основні програми, так і функції мови Python. Сама система теж написана на мові Python і генерує логічні умови коректності та завершимості трас в програмах методом Флойда–Хоора [Fl].

Система *VerPro* зводить проблему верифікації програм, анотованих інваріантами та варіантами, до проблеми перевірки тотожної істинності формул в деякій формальній логічній системі. Система *VerPro* має власні засоби перевірки таких логічних формул шляхом спрощенням їх за допомогою еквівалентних перетворень.

Однак, цей підхід виявився доволі обмеженим. Тому для перевірки логічних умов у новій версії системи *VerPro* використано механізм солверів популярної системи доведення теорем *Z3* [Z3]. В арифметиці цілих чисел без структур даних система *Z3* справляється практично з усіма логічними виразами, які ми можемо придумати.

За допомогою солверів *Z3* система *VerPro* успішно верифікує правильно анотовані програми. Якщо ж певна логічна формула виявляється не тотожно істинною, тоді *Z3* знаходить контрприклад для гіпотези про тотожну істинність формули.

Ідея пошуку контрприкладів при невдалій верифікації програми розглядається вперше, як і застосування для верифікації програм системи доведення теорем Z3.

В даній роботі описано роботу останньої версії системи. Нова версія системи *VerPro* реалізує віконний діалоговий інтерфейс [K2]. Для цього було використано розширення PyQt5 [Qt] інтерпретатора мови Python. Нова версія системи також дозволяє знаходити контрприклади для логічних умов, які не вдалося довести.

Основні результати даної статті такі: описано інтерфейс системи *VerPro*, яка дозволяє верифікувати програми з маленької підмножини сучасної мови програмування Python; для перевірки умов верифікації застосовано одна з найпотужніших систем доведення теорем Z3; реалізовано механізм для пошуку контрприкладів у випадку невдалої верифікації [z3py].

При розширенні цієї області на багатосортні алгебри та структури даних ситуація кардинально змінюється. У формулах з'являються квантори і навіть прості та зрозумілі функції описуються складними виразами. Наприклад, достатньо розглянути функцію *LeftPad* [LP]. Іншу проблему становить побудова інваріантів та варіантів програм. Дальший розвиток системи *VerPro* ведеться в напрямку подолання цих проблем.

Перспективи використання методу верифікації для встановлення функціональних властивостей програм активно обговорюються в наукових публікаціях [Br, St, Va].

СТРУКТУРА СИСТЕМИ VERPRO

Система *VerPro* складається з двох основних підсистем: підсистеми генерації умов коректності та підсистеми перевірки їх тотожної істинності. Для генерації умов коректності застосовується технологія символьного виконання.

Для перевірки умов верифікації застосовується механізм еквівалентного перетворення формул згідно відомих співвідношень з області арифметики та математичної логіки.

Для перевірки умов верифікації в системі *VerPro* було реалізовано три бібліотеки класів та функцій:
- *arilib* – бібліотека арифметичних функцій;
- *logilib* – бібліотека логічних функцій;
- *treelib* – бібліотека функцій над деревами.

Основні класи, які реалізовані в цих бібліотеках:
- *Tree* – клас деревовидних структур, який представляє вирази;
- *Monom* – клас одночленів, який представляє стандартизовані одночлени;
- *Polynom* – клас поліномів, який представляє стандартизовані поліноми;
- *Relation* – клас відношень, який представляє стандартизовані відношення поліномів;
- *Conjunct* – клас кон'юнкцій, який представляє стандартизовані кон'юнкції відношень;
- *Implication* – клас імплікацій, який представляє умови коректності.

Для посилення засобів перевірки умов верифікації програм у новій версії системи було реалізовано механізм солверів системи доведення теорем Z3 [Z3, Z3py].

У новій версії системи окрім україномовного реалізовано також англомовний інтерфейс. Він включає назви діалогових елементів форм, надписи та повідомлення, спливаючі підказки тощо. Для переключення інтерфейсів, а також для налаштування параметрів системи застосовуються файли формату JSON.

Специфіка вхідної мови розширення *Z3py* та мови Python обумовила механізм реалізації підсистеми перевірки умов коректності трас програми. Ця підсистема реалізована за допомогою трьох основних програм:
- Програма *winMain*;
- Програма *winCheckZ3*;
- Програма *counterExample*.

Програма *winMain* генерує умови верифікації трас та спрощує їх застосуванням еквівалентних перетворень. Умови верифікації представляють собою вирази з логічними та арифметичними операціями, всі параметри яких є вільними змінними. Для використання солверів *Z3* програма *winMain* генерує спеціальний модуль *z3cond* з описом функції *checkcondsz3* на вхідній мові розширення *Z3py*.

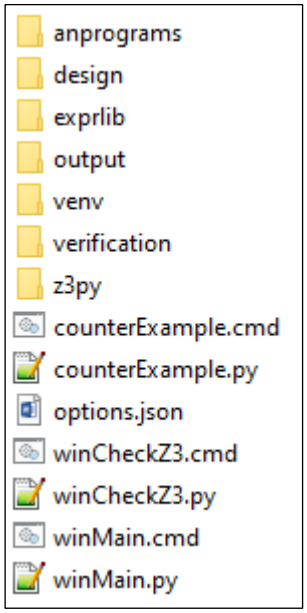

Рис. 1. Структура каталогу проекту

Програма *winCheckZ3* викликає згенеровану функцію *checkcondsz3* з умовами верифікації, які не вдалося вивести програмі *winMain*. Функція checkcondsz3 для перевірки умов верифікації застосовує солвери системи *Z3* [Z3]. Для кожної умови верифікації, тотожну істинність якої не вдалося вивести за допомогою солвера *Z3*, користувач зможе згенерувати контрприклад. Для цього програма генерує спеціальний модуль *z3counter*.

Програма *counterExample* викликає функції, опис яких розташовано в модулі *z3counter*, генеруючи контрприклад для умови, тотожну істинність якої не вдалося встановити ні програмі *winMain*, ні програмі *winCheckZ3*.

Рис. 1 демонструє структуру системи *VerPro*. В каталозі проекту розташовуються модулі основних програм та командні файли для їх виклику.

В підкаталозі *anprograms* знаходяться модулі анотованих програм. В підкаталозі *design* знаходяться ui–файли з описом віконного інтерфейсу програм системи та модулі її інтерфейсних класів.

Підкаталог *exprlib* містить згадані вище бібліотеки *arilib*, *logilib* та *treelib*. Підкаталог *z3py* містить модулі, призначені для взаємодії з розширенням *Z3py*.

Підкаталог *output* містить текстові файли зі звітами про етапи перетворення анотованої програми та її умов верифікації.

Підкаталог *verification* задає бібліотеку функцій, призначених для реалізації задач верифікації програм, написаних мовою Python.

Підкаталог *venv* – це підкаталог віртуального оточення проекту, яке включає розширення *PyQt5* та *Z3py*.

ІНТЕРФЕЙС СИСТЕМИ VERPRO

Основна програма *winMain* для утворення інтерфейсу викликає модуль *verify* з описом інтерфейсного класу. Функціонування системи та її інтерфейс продемонструємо на прикладі функції *frac* обчислення частки та залишку від ділення двох цілих додатних чисел [Fl].

Інтерфейсний файл *verify.ui* дизайнера розширення *PyQt5* визначає вікно форми *Verify*, у якому відображаються таблиці з результатами аналізу. На верхній панелі вікна відображаються кнопки, які стають доступними у відповідний момент аналізу. На нижній панелі можуть відображатися повідомлення користувачу.

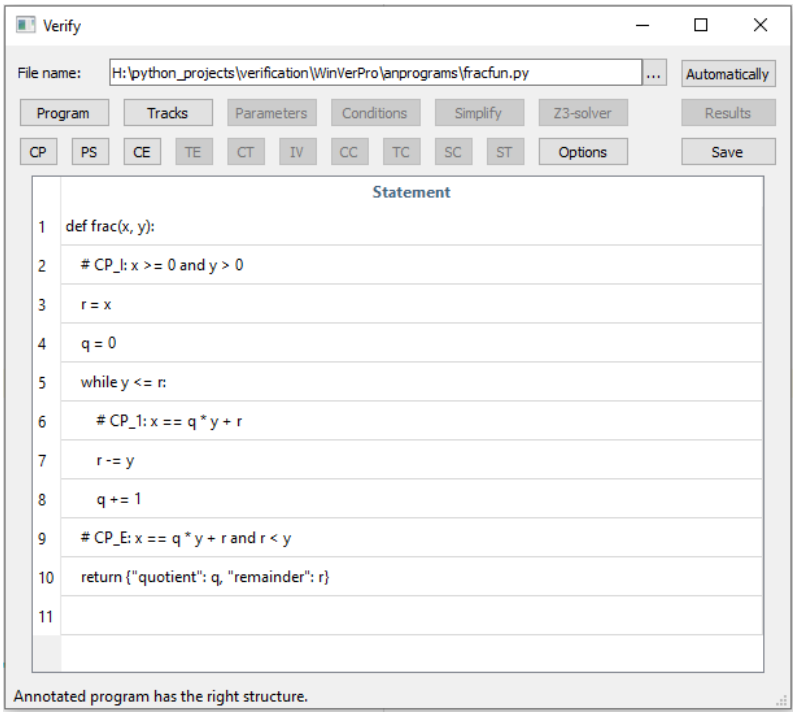

Рис. 2. Інтерфейс програми winMain

Поле вводу *File name* дозволяє вибрати програму для аналізу. Кнопка *Program* аналізує її та відображає в таблиці (рис. 2). Кнопка *CP* відображає таблицю з контрольними точками програми. До них відносяться точка входу програми та точка виходу з неї, а також точки циклів програми (рис. 3a).

	Program CP	Type
1	CP_I	Initial
2	CP_1	Loop
3	CP_E	Ending

a)

	CP	Condition
1	CP_I	x >= 0 and y > 0
2	CP_1	x == q * y + r
3	CP_E	x == q * y + r and r < y

b)

Рис. 3. Контрольні точки програми та умови в них

Кнопка *PS* відображає в таблиці детальну структуру програми, яка полегшує розуміння помилок у структурі програми та її анотації

Кнопка *CE* відображає умови, асоційовані з контрольними точками програми, до яких відносяться початкова, кінцева умови та інваріанти циклів (рис. 3b).

Кнопка *Tracks* будує траси в програмі, які ведуть від однієї контрольної точки до іншої. А кнопка *TC* відображає траси та їх оператори й умови переходу (рис. 4).

	Track	Statements
1	CP_I -> CP_1:	r = x, q = 0, y <= r
2	CP_I -> CP_E:	r = x, q = 0, y > r
3	CP_1 -> CP_1:	r -= y, q += 1, y <= r
4	CP_1 -> CP_E:	r -= y, q += 1, y > r

Рис. 4. Траси та їх оператори

Кнопка *Parameters* визначає необхідні символьні параметри для символьного виконання трас програми. Якщо значення змінної на трасі не змінюється, тоді вона сама буде позначати своє значення. Кнопка *IV* видає символьні позначення змінних для кожної траси програми (рис. 5).

	Track	Initialization of variables
1	CP_I -> CP_1:	
2	CP_I -> CP_E:	
3	CP_1 -> CP_1:	r = r0, q = q0
4	CP_1 -> CP_E:	r = r0, q = q0

Рис. 5. Параметри трас

	Track	Correctness condition
1	CP_I -> CP_1:	x >= 0 and y > 0 and x - y >= 0 -> x - x == 0
2	CP_I -> CP_E:	x >= 0 and y > 0 and -x + y > 0 -> x - x == 0 and -x + y > 0
3	CP_1 -> CP_1:	-q0 * y - r0 + x == 0 and r0 - y - y >= 0 -> -q0 * y - r0 + x - y + y == 0
4	CP_1 -> CP_E:	-q0 * y - r0 + x == 0 and -r0 + y + y > 0 -> -q0 * y - r0 + x - y + y == 0 and -r0 + y + y > 0

Рис. 6. Умови коректності трас

Кнопка *Conditions* будує умови коректності трас програми, а кнопка *CC* відображає їх у таблиці (рис. 6).

Кнопка *Simplify* спрощує побудовані умови застосуванням до них еквівалентних перетворень. А кнопка *SC* відображає спрощені умови в таблиці (рис. 7).

	Track	Simplified correctness condition
1	CP_I -> CP_1:	True
2	CP_I -> CP_E:	True
3	CP_1 -> CP_1:	True
4	CP_1 -> CP_E:	True

Рис. 7. Спрощені умови коректності трас

Кнопка *Results* видає результат спрощення умов верифікації: "The ptogram is correct". Таким чином, коректність функції *frac* системі *VerPro* вдалося вивести застосуванням еквівалентних перетворень. Кнопка *Automatically* виконує всі описані дії автоматично, відображаючи в таблиці лише загальний результат.

Розгляд властивостей програми доповнимо властивістю завершимості. Інваріант та варіант циклу візьмемо близькими до наведених в роботі [Fl]. Для цього в контрольній точці циклу побудуємо такий коментар:

CP_1: x == q * y + r and x - q >= 0; x – q .

На цей раз еквівалентні перетворення системи *VerPro* успішно справилися з умовою завершимості, але не справилися з умовою коректності третьої траси. Кнопка *Results* видає повідомлення: "Failed to prove 1 correctness condition of program tracks".

	Track	
1	CP_I -> CP_1:	True
2	CP_I -> CP_E:	True
3	CP_1 -> CP_1:	q0 * y + r0 - x == 0 and -q0 + x >= 0 and r0 - 2 * y >= 0 -> -1 - q0 + x >= 0
4	CP_1 -> CP_E:	True

Рис. 8. Спрощені умови коректності трас

Кнопка *SC* видасть спрощені умови коректності трас (рис. 8). Оскільки не всі умови коректності системі VerPro вдалося перевірити власними засобами, вона робить доступною кнопку *z3-solver*. Ця кнопка утворює модуль *z3cond* і розташовує його в підкаталозі *z3py*. В таблиці з`являються два рядки повідомлень про успішне утворення модуля виклику розширення *Z3py* (рис. 9).

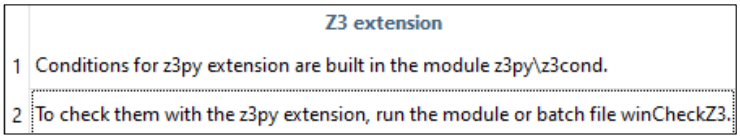

Рис. 9. Повідомлення про утворення модуля z3cond

ПЕРЕВІРКА ТОТОЖНЬОЇ ІСТИННОСТІ ЗА ДОПОМОГОЮ Z3PY

Для перевірки тотожної істинності умов, згенерованих в модулі *z3cond*, викличемо програму *winCheckZ3*. Програма видасть вікно з такими 5 кнопками меню та таблицею з умовою коректності траси номер 3 (рис. 10).

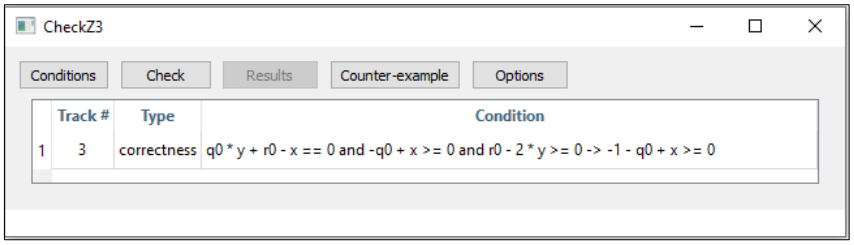

Рис. 10. Вікно програми winCheckZ3

Кнопка *Conditions* відображає умови верифікації, які не вдалося вивести застосуванням еквівалентних перетворень. Кнопка *Check* визначає (рис. 11), чи є умови верифікації тотожно істинними (*True*), чи ні (False).

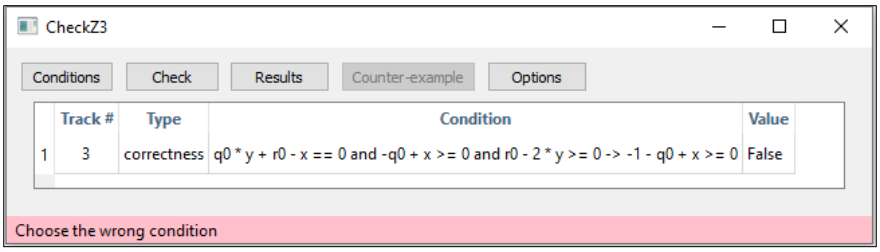

Рис. 11. Перевірка тотожної істинності умов верифікації

Якщо якась умова не є тотожно істинною, тоді її можна виділити і побудувати контрприклад – набір значень параметрів, який робить її фальшивою. Панель статусу відображає рекомендації про це користувачу (рис. 12).

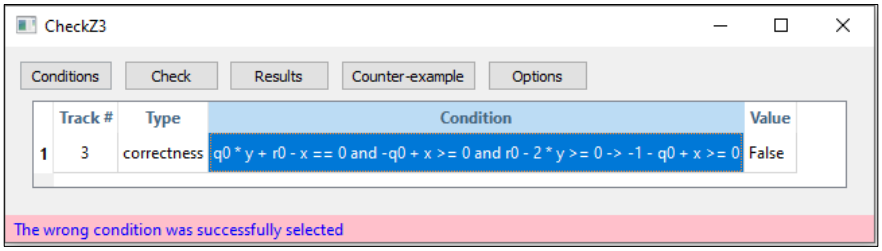

Рис. 12. Виділення сумнівної умови

Для побудови прикладу потрібно натиснути на кнопку *Counter–example*, яка в підкаталозі *Z3py* утворить модуль *z3Counter*. Цей модуль містить опис двох функцій: *z3countP* та *z3counterEx*.

Для виклику цих функцій потрібно запустити програму *counterExample*. Ця програма теж утворює кнопочне меню та відображає таблицю (рис. 13).

Кнопка *Condition* відображає в таблиці умову та пояснює її походження. Кнопка *Build* знаходить набір значень параметрів, які роблять її фальшивою (рис. 14).

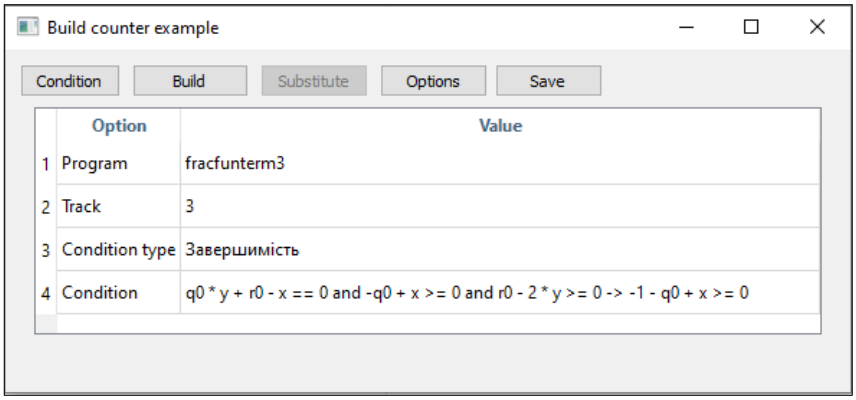

Рис. 13. Вікно програми *counterExample*

	Parameter	Value
1	Condition	False
2	x	2
3	r0	6
4	y	-2
5	q0	2

Рис. 14. Значення параметрів у контрприкладі

	Expression	Value
1	Program	fracfunterm3
2	Track number	3
3	Correctness condition	q0 * y + r0 - x == 0 and -q0 + x >= 0 and r0 - 2 * y >= 0 -> -1 - q0 + x >= 0
4	Parameter values	x = 2, r0 = 6, y = -2, q0 = 2
5	The result of the substitution	-2 * 2 - 2 + 6 == 0 and -2 + 2 >= 0 and -2 * -2 + 6 >= 0 -> -1 - 2 + 2 >= 0
6	Calculation of polynomials	0 == 0 and 0 >= 0 and 10 >= 0 -> -1 >= 0
7	Calculation of polynomials	True and True and True -> False
8	Calculation of implication	True -> False
9	Value of the condition	False

Рис. 15. Покрокове обчислення значення умови

Кнопка *Substitute* підставляє знайдені параметри в умову і покроково обчислює її значення (рис. 15).

Причиною невдачі верифікації є помилкові інваріант та варіант циклу. В ролі варіанта візьмемо вираз *r*, а інваріант (рис. 2) доповнимо кон'юнктом y > 0, який допоможе забезпечити скінченість послідовності значень виразу. В результаті коментар з умовою в контрольній точці циклу прийме такий вигляд:

CP_1: x == q * y + r and y > 0; r .

Кнопка TC видасть таку умову завершимості траси 3:

*-q0 * y - r0 + x == 0 and y > 0 and r0 - y - y >= 0 -> y > 0*

Кнопка *Simplify* спростить побудовану умову до тотожної істини, а кнопка *Results* відобразить такий результат спрощення умов верифікації: "The program is totally correct".

Список використаних джерел:

1. Костирко В.С. Система верифікації python–програм. Матеріали XXV Міжнародної науково-практичній конференції. К.: Європейський університет. 2019. С. 75–78.

2. Floyd, R. W. Assigning meanings to programs. Proceedings of a Symposium on Applied Mathematics. American Mathematical Society. 19 Mathematical Aspects of Computer Science. 1967. 19–31.
3. Z3 API in Python. Retrieved from: https://ericpony.github.io/z3py-tutorial/guide-examples.htm.
4. Костирко, В. С., Плеша, В. І. (2021). Система верифікації програм VerPro. Сучасні напрями розвитку економіки, підприємництва, технологій та їх правового забезпечення: матеріали Міжнародної науково-практичної конференції. Львів: Львівський торговельно-економічний університет. 287–288.
5. PyQt5 Reference Guide. Retrieved from: https://www.riverbankcomputing.com/ static/Docs/PyQt5/.
6. Костирко, В.С., Плеша, В. І. Застосування бібліотеки Z3py для перевірки умов коректності та завершимості програм. Матеріали XXVI міжнародної науково-практичної інтернет–конференції. К.: Європейський університет. 2020. 80–83.
7. Wayne, H. Retrieved from: https://github.com/hwayne/lets-prove-leftpad.
8. Bruni, R., Giacobazzi, R., Gori, R., Ranzato, F. (2023). A Correctness and Incorrectness Program Logic. Journal of the ACM. 70. 1–45. DOI: https://doi.org/10.1145/3582267.
9. Stump, A. (2016). Verified Functional Programming in Agda. Association for Computing Machinery and Morgan & Claypool. DOI: https://doi.org/10.1145/2841316.
10. Vardi, M. Y. (2021). Program verification: vision and reality. Communications of the ACM (CACM). 64 (7). 5. DOI: https://doi.org/10.1145/3469113.

References:

1. Kostyrko, V. S., Plesha, V. I. (2019). Systema veryfikacii python–program. Система верифікації python–програм [The Verification System for Python programs]. Materialy XXV Mizhnarodnoyi naukovo–praktychnoyi konferenciyi. K.: Yevropeiskyj universytet. 75–78. [in Ukrainian] [K1]
2. Floyd, R. W. (1967). Assigning meanings to programs. Proceedings of a Symposium on Applied Mathematics. American Mathematical Society. 19: Mathematical Aspects of Computer Science. 19–31. [in English] [Fl]
3. Z3 API in Python. Retrieved from: https://ericpony.github.io/z3py-tutorial/guide-examples.htm. [in English] [Z3]
4. Kostyrko, V. S. (2021). Systema veryfikaciyi prohram VerPro [The program verification system VerPro]. Suchasni napryamy rozvytku ekonomiky, pidpryiemnyctva, tekhnolohij ta yikh pravovoho zabezpechennya: materialy Mizhnarodnoyi naukovo-praktychnoyi konferenciyi. Lviv: Lvivskyj torhovelno–ekonomichnyj universytet. 287–288. [in Ukrainian] [K2]
5. PyQt5 Reference Guide. Retrieved from: https://www.riverbankcomputing.com/static/Docs/PyQt5/. [in English] [Qt]
6. Kostyrko, V. S., Plesha, V. I. (2020). Zastosuvannya biblioteky Z3py dlya perevirky umov korektnosti ta zavershymosti prohram [Application of the Z3py library to check the program correctness and termination conditions]. Materialy XXVI mizhnarodnoyi naukovo–praktychnoyi internet–konferenciyi. K.: Yevropejskyj universytet. 80–83. [in Ukrainian] [z3py]
7. Wayne, H. Retrieved from: https://github.com/hwayne/lets-prove-leftpad. [in English] [LP]
8. Bruni, R., Giacobazzi, R., Gori, R., Ranzato, F. (2023). A Correctness and Incorrectness Program Logic. Journal of the ACM. 70. 1–45. DOI: https://doi.org/10.1145/3582267. [in English] [Br]
9. Stump, A. (2016). Verified Functional Programming in Agda. Association for Computing Machinery and Morgan & Claypool. DOI: https://doi.org/10.1145/2841316. [in English] [St]
10. Vardi, M. Y. (2021). Program verification: vision and reality. Communications of the ACM (CACM). 64 (7). 5. DOI: https://doi.org/10.1145/3469113. [in English] [Va]