https://doi.org/10.31649/1997-9266-2023-167-2-72-80

УДК 681.3.06

І. В. Редько[1]
М. О. Зилевіч[1]

ТЕОРЕТИЧНІ ОСНОВИ ПРОГРАМНОЇ РЕЛЯТИВІЗАЦІЇ У ТЕХНОЛОГІЧНИХ СИСТЕМАХ ПРОГРАМУВАННЯ

[1]Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

У дослідженні аргументовано необхідність парадигмальних змін у технологічних основах програмування, які передбачають перехід від індивідно-суб'єктивної парадигми програмування до інтерсуб'єктивної. Запропонована інтерсуб'єктивна парадигма базується на розумінні програмування як діяльності, що обумовлена програмним уподібненням, суб'єкто-орієнтованим взаємодоповненням його активної та пасивної форм. Для реалізації такого переходу, розглянуто об'єктивізацію активно-пасивного взаємодоповнення, яка є основною передумовою реального осучаснення розуміння програмування як рефлексивно-транзитивного замикання породжуваного суб'єктом програмного уподібнення. Визначальною для об'єктивізації активної ролі суб'єкта в контексті технологічних середовищ програмування є концепт — суть–уподібнення, представлений у вигляді тієї чи іншої специфікації. Подобою ж як цілим наслідком уподібнення, тобто програмою об'єктивізується пасивна складова обумовлення. Для досягнення цієї мети, запропоновано інструмент логіко-математичних специфікацій семантико-синтаксичних аспектів програмування. Цей інструмент апробовано на прикладах, які продемонстрували загальні особливості застосування технологічного середовища програмування до породження суб'єкто-орієнтованих технологічних систем програмування та їхнього використання. Зокрема, застосування цього інструменту дозволяє створювати ефективніші технологічні системи програмування, які мають вищу якість та гнучкість у виконанні завдань. Отримані результати підтверджують необхідності парадигмальних змін у технологічних основах програмування та є одним з можливих варіантів розв'язання проблеми парадигмальних змін і подальшої технологізації програмування, а також демонструє загальні особливості застосування технологічного середовища програмування для породження суб'єктно-орієнтованих технологічних систем програмування та їхнього використання.

Ключові слова: концепт, монада, композит, композиція, редукція, середовище програмування, оракульні схеми.

Вступ

Відомо, що свідоме вирішення будь-якого завдання передбачає реалізацію певного плану, послідовне виконання якого є формальною запорукою досягнення мети. Сьогодні такий план асоціюється з розумінням розв'язання основної проблеми як інтеграції підзадач. Способи отримання та формулювання таких рішень залежать від предметної області, складності завдання, ресурсів для реалізації, рівня кваліфікації виконавця. Це суттєво впливає на якість отриманих результатів, їхню обґрунтованість і правдивість, ефективність формулювання розв'язків, можливість їхнього повторного використання в нових задачах.

Донедавна парадигма «розділяй і володарюй» у програмуванні давала виняткові результати, які говорили самі за себе, ігноруючи серйозні труднощі, які виникали та продовжують існувати в процесі. Цей негативний аспект сприймався як об'єктивна проблема зростання. Однак на початку свого розвитку цей підхід навіть сприяв прогресу в програмуванні, надаючи якомога ширшу фактографію програмування як основу для розвитку фактології програмування. Прикладом цього є відома енциклопедична монографія Д. Кнута «Мистецтво програмування» [1].

Враховуючи реалії сьогодення, те, що колись можна було проігнорувати як менш важливе, згодом стає ключовим. Якщо розглядати програму як продукт, то вона повинна базуватися на технологічних принципах обробки інформації та процесів, пов'язаних з такими видами робіт. Але, з

іншого боку, реалізація програми продовжує бути надто персоніфікованою, оцінюватись лише за кінцевим результатом і бути нерозривно пов'язаною з її виконавцем. Такий підхід сприяє максимальній свободі творчості суб'єктів програмування та сприймає програмування як мистецтво, наголошуючи на фактографічній складовій з усіма властивостями, які супроводжують такий підхід. Головний недолік такого підходу це надто спрощене розуміння програмування, яке не відповідає сучасним вимогам. Це проявляється в тому, що, як і в будь-якому мистецтві, в програмуванні акцент зазвичай робиться на результат творчого процесу — програмі, і занадто мало уваги приділяється самому процесу розробки, який залишається повністю під відповідальністю програміста. У результаті неможливо не тільки вирішити, але й серйозно розглянути питання дослідження продуктивних засад програмування. Оскільки базові властивості програм формуються саме на етапі їхньої розробки, таке спрощення може стати перешкодою для розробки реальних технологічних основ програмування. Тому важливо збагатити розуміння програмування.

Дослідженнями [2]—[4] доведено, що для розв'язання цієї проблеми необхідні радикальні зміни підходу до програмування — перехід від індивідуально-суб'єктної парадигми до інтерсуб'єктивної, яка враховує активну роль суб'єкта в програмуванні. В основі цього переходу лежить суть сутнісне уподібнення (ССУ) — особливий тип зв'язку, що виникає внаслідок релятивізації причинно-наслідкового зв'язку між процесом розв'язання проблеми та її наслідком [2], [3]. Тому програмна релятивізація (ПР) як суттєва релятивізація програмування, спрямована на його технологізацію, є *основною темою цієї роботи*. Роботи [3], [4] демонструють, що платформа дослідження релятивізації програм є абстрактною, тобто забезпечує взаємодію двох об'єктивно не пов'язаних типів абстракції, де замкнута інтерсуб'єктивна логіка є ядром технологічного середовища програмування (ТСП), а відкритий різновид суб'єкт-орієнтованих продовжень — технологічним середовищем програмування (ТСРП), які активно-пасивно доповнюють одна одну. Основною *метою дослідження* є вивчення взаємодії між замкнутою інтерсуб'єктивною логікою ТСРП і відкритим різновидом суб'єкт-орієнтованих продовжень ТСП, що ґрунтується на їхньому взаємному доповненні. Цей взаємодоповнюючий зв'язок забезпечується істотними аналогіями, спрямованими на врахування активної ролі суб'єкта програмування. Також, однією з основних цілей дослідження є об'єктивізація суб'єкта програмування через актуалізацію інтерсуб'єктивних модальностей, властивих ТСРП як методу програмної релятивізації, а також розвиток і практичне застосування цієї взаємодії.

Програмне середовище як інтерсуб'єктивна платформа для програмної релятивізації

Зі сказаного випливає, що для досягнення поставленої мети необхідно розширити поняття «система програмування» цілеспрямованим аспектом логіко-предметної взаємодоповнюваності ТСРП і ТСП. Основою інтерсуб'єктивної парадигми є розуміння програмування як строго визначеної та формалізованої діяльності. Розвиток цієї парадигми, включаючи ТСП як платформу для релятивізації програм, є поступовим збагаченням цієї передумови. Цей процес рухається в напрямку «від цілісного представлення до продуктивного розуміння» програмування, де продуктивність означає технологізацію.

Для того, щоб продуктивно збагатити цілісне розуміння програмування, необхідно спочатку детально зрозуміти чинну термінологію. Це означає, що збагачення має бути достатньо обґрунтованим, відповідно до принципу достатньої підстави. У цьому контексті важливо розуміти термін «програма» як уподібнення суттєвої риси [5], [6]. Це дозволяє продуктивно збагатити розуміння програмування як взаємодії двох типів абстракцій: модальної та реальної. Розуміння програмування як доповнення до цих двох типів абстракцій дозволяє збагатити розуміння програмування як діяльності, керованої програмним уподібненням (ПУ). Таким чином, отримуємо продуктивне збагачення вихідного основоположення: програмування — діяльність, обумовлена ПУ.

Подальший крок у розвитку ТСРП пов'язаний з продуктивним збагаченням програмних аналогій, головною особливістю якого є неможливість об'єктивного узагальнення модальності сутності, тобто створення подібної сутності, та реальної сутності, яка оновлюється програмою. З цього безпосередньо випливає біабстрактність ТСРП як носія програмних уподібнень.

Відповідно до вищевикладеного кожна ПУ виникає на основі обумовленості сутності шляхом асоціації його з якоюсь умовою. Оскільки такі умови може забезпечити лише суб'єкт обумовлення як засіб об'єктивації, то порівняння є реалізацією взаємодоповнюваності активної та пасивної форм обумовлення. Тому для реального розширення розуміння програмування як рефлексивно-

транзитивного замикання суб'єктивної програмної подібності необхідна об'єктивація активно-пасивної комплементарності. Визначальною для об'єктивізації активної ролі суб'єкта в контексті побудов у ТСРП є концепт — суть – уподібнення, представлена у вигляді тієї чи іншої специфікації. Подобою ж як цілим (єдиним, монадним) наслідком уподібнення, тобто — програмою у наведеному вище сенсі — об'єктивізується пасивна складова обумовлення. Таким чином, дійшли до ще одного експлікаційного збагачення ПУ, де поняття є сутністю, що зумовлює сутність, а програма є сутністю, зумовленою поняттям.

Концепт забезпечує цілісність побудов, а програма — їхню продуктивність. Перехід забезпечується через ПУ. Слід підкреслити, що такі терміни, як поняття, програма та умови, вводяться як абстракції об'єктивних умов, а не конкретних ознак об'ємів, що дозволяє використовувати їх як оракули. Отже, наведена концептопрограмна дефініція є відкрито-замкнутим оракульним середовищем, яке є цілісним за своєю природою [2], [4]. Однак для досягнення продуктивніших результатів необхідно ще більше збагачувати це середовище.

Розгляд наведених середовищ веде до розгляду оракулів вищого рівня, які є джерелом продуктивності побудов. Серед них особливе місце займають композиційні схеми, а саме загальні композиційні схеми та базові композиційні схеми [2]. Для зручності будемо використовувати терміни «композиції» і «композити». Дослідження композиційного програмування [4], [7]—[13], переконливо доводять, що таке композиційно-композиційне збагачення запропонованої парадигми програмування є лейбніцевим. Це означає, що ТСРП, як реалізацію цілісного розуміння програмування, можна звести до композиційно-композитного збагачення заданого визначення концептуального програмування. Тому всі подальші збагачення носять вже не цілісний, а продуктивний, орієнтований на технологізацію, характер і пов'язані з суб'єктно-орієнтованими замиканнями ТСРП до відповідних ТСП.

Аналіз логічних та предметних передумов програмної релятивізації

З вищезазначеного випливає, що ТСП може бути створена лише шляхом програмного уподібнення, яке відбувається покроково в контексті концептопрограмного замикання ТСРП, що є відкрито-замкненим композитологічним оракульним середовищем. Це досягається за допомогою активації відповідних оракулів. Таким чином, метод програмної релятивізації (МПР) базується на концептопрограмному замиканні та є засобом для специфікації ТСП. Отримувана ТСП є системою, орієнтованою на суб'єкта, засобом, що дозволяє суб'єкту активно втілюватись в програмуванні. Її інтенсіональний характер є результатом уточнень базових та похідних генетичних структур як концептів, притаманних як суб'єкту, так і об'єкту програмування, базових предметних операцій та композито-композиційних інтерфейсів.

Розглянемо ці кроки детальніше, з фокусом на прагматиці задачі. Для актуалізації ТСРП за допомогою композито-композиційного підходу, будемо використовувати апарат примітивної програмної алгебри (ППА) [14], чия сигнатурна операційна система буде складовою частиною програмного уподібнення створюваної ТСП. Як приклад предметної основи побудови ТСП, виберемо арифметичні перетворення, що будуть уточнені як обчислювальні багатомісні арифметичні функції та предикати. Розглянемо особливості створення ТСП на прикладах програмування арифметичних перетворень. Ці приклади мають на меті продемонструвати загальні особливості використання ТСРП для прагматико-обумовленого створення та використання ТСП. Тому, зазначені вище предметні домовленості не впливають на репрезентативність побудов.

Носій арифметичної ППА складають *n*-арні функції та предикати виду $N^n \to N$ та $N^n \to \{T, F\}$, $n \in N$ відповідно [15]. Сигнатуру ППА складають операції суперпозиції, розгалуження і циклування, що є адекватними уточненнями основних методів конструювання програм [8]—[12]. У акцентуванні уваги на генетичних особливостях розглядуваних функцій та предикатів у їхньому позначенні перевага надаватиметься операторній, а у акцентуванні на результатах застосування композицій — термальним формам запису [15].

Нехай задані m функцій однакової арності, наприклад, k — $f_1,\ldots, f_m$, m-арна функція f та n-арний предикат $h, m, k \in N$. Розглянемо ще нові k-арні функції g, d, c, значення яких на аргументі $a_1,...,a_k$ задаються так:

$$g(a_1,...,a_k) \cong f\left(f_1(a_1,...,a_k),\ldots, f_m(a_1,...,a_k)\right);$$

$$d(a_1,...,a_k) \cong \begin{cases} f_1(a_1,...,a_k), \text{якщо } h(a_1,...,a_k) = T, \\ f_2(a_1,...,a_k), \text{якщо } h(a_1,...,a_k) = F, \end{cases}$$

$c(a_1,...,a_k) \cong a_1^j$, де a_1^j — перша компонента першого кортежу послідовності кортежів $\left[a_1^i,...,a_k^i\right]_{i=1,2,...}$, де $a_s^1 = a_s$, $s = 1,...,k$, $a_s^{i+1} = f_s\left(a_1^i,...,a_k^i\right)$, для якого $h\left(a_1^j,...,a_k^j = F\right)$, за умови, що для всіх $r = 1,..., j-1$ значення $h\left(a_1^r,...,a_k^r = T\right)$, ($\cong$ — розуміється як умовна рівність).

Вважатимемо, що функція g є результатом застосування композиту $m+1$-арної суперпозиції S^{m+1} до кортежу функцій $(f, f_1,..., f_m)$, функція d-композиту тернарної операції розгалуження $\Diamond$ до кортежу функцій (h, f_1, f_2), а c-композиту $k+1$-арного циклування до кортежу $(h, f_1,..., f_k)$. Тобто, $g \equiv S^{m+1}(f, f_1,..., f_m)$, $d \equiv \Diamond(h, f_1, f_2)$ та $c \equiv *^{k+1}(h, f_1,..., f_k)$.

Замикання множини функцій та предикатів σ операціями ППА, 0^1 (0-функцію), що будь-якому натуральному числу ставить у відповідність 0, тобто: $S^2\left(0^1, I_1^1\right)(n) = 0|_{n\in N}$, s^1 — функцію слідування: $S^2\left(s^1, I_1^1\right)(n) = n+1|_{n\in N}$, $+^2$ — суму двох натуральних чисел: $S^3\left(+^2, I_1^2, I_2^2\right)(n,m) = n+m|_{n,m\in N}$, $\times^2$ — добуток двох натуральних чисел: $S^3\left(\times^2, I_1^2, I_2^2\right)(n,m) = n\times m|_{n,m\in N}$, $<^2$ — предикат «менше»: $S^3\left(<^2, I_1^2, I_2^2\right)(n,m) = \begin{cases} T, \text{ якщо } n < m, \\ F, \text{ якщо } m \le n \end{cases}|_{n,m\in N}$ та I_n^m — селекторну функцію:

$$I_n^m(a_1,\ldots,a_m)|_{m,n\in N} = \begin{cases} a_n, \text{ якщо } n \le m, \\ \perp, \text{ інакше.} \end{cases}$$

Відповідно носій ППА виглядатиме так: $\left[0^1, s^1, +^2, \times^2, <^2, I_n^m\right]_\Omega = £$ [16]. Цей результат разом з вибором операцій з сигнатур у якості актуалізації базових композитів ТСРП забезпечує логіко-предметну основу для продуктивної специфікації арифметичної ТСП. Особливо важливим тут є інтерфейс між композитами з сигнатур ППА та похідними від них композиціями. Обумовлені композитами редукційні структури, як показано в [4] відіграють ключову роль у побудові таких інтерфейсів. Це забезпечує головну особливість створюваних таким чином ТСП — вони реально, а не лише номінально, підтримують причинно-наслідкове взаємодоповнення двох складових вирішення будь-якої програмістської задачі — програмування як породження та застосування композицій та програми як наслідку програмування.

Редукційні підходи до програмної релятивізації

Сигнатури ППА можна переформулювати як концепт ТСРП, що дозволяє зосередитись на об'єктивізації ролі суб'єкта програмування арифметичної ТСП. Це відкриває можливості для використання логіко-математичних специфікацій і семантико-синтаксичних аспектів програмування арифметичних задач у дослідженні ТСРП.

Розв'язання будь-якої задачі є наслідком об'єднання рішень її підзадач [17]. У процесі цього об'єднання роль суб'єкта є важливою. Якщо задача є простою, то цей процес зазвичай не потребує виділення суб'єкта окремо. Але у випадку складної задачі, процес об'єднання рішень стає головним аспектом, оскільки він визначає складність задачі. У такому випадку побудова рішення вимагає використання двох типів логіко-математичних специфікацій — специфікацій підзадач та специфікацій їхнього об'єднання. У випадку розглянутої ТСП засобами об'єднання є генетичні структури, які є похідними від згаданих генних структур або композитів. Специфікації підзадач базуються на багатомісних функціональних та декомпозиційних структурах. Останні базуються на спеціалізаціях редукційних схем, які введені в джерелі [4] та обмежені базисними композитами. Розглянемо їх, починаючи зі схем, обумовлених композитами суперпозиції S^n, $n = 2,3,\ldots$ та розгалуження $\Diamond$.

Кортеж функцій $(g_1,..., g_m)$, де $g_1,..., g_m$ — функції виду $N^k \to N|_{k\in N}$, називається S^{m+1}-редук-

цією *k*-арної функції f, якщо існує така функція $g: N^m \to N|_{m\in N}$, що $f = S^{m+1}(g, g_1, ..., g_m)$.

Кортеж *n*-арних функцій (g_1, g_2) називається $\Diamond$-редукцією *n*-арної функції f якщо існує *n*-арний предикат *h* такий, що $f = (\Diamond(h, g_1, g_2))$.

Безпосередньо з наведених визначень випливають прості та корисні необхідні ознаки S^{m+1}- та $\Diamond$-редукційності. Називатимемо рангом кортежу функцій $(g_1, ..., g_m)$ множину $r(g_1, ..., g_m) \equiv \left\{ g_1(a_1, \ldots, a_k), \ldots, g_m(a_1, \ldots, a_k) \middle| g_i(a_1, \ldots, a_k) \in ran(g_i)|_{i=\overline{1,m}} \;\&\; (a_1, \ldots, a_k) \in \bigcap_{i=\overline{1,m}} dom(g_i) \right\}$. Тоді, якщо $(g_1, ..., g_m)$ є S^{m+1}-редукцією функції f, то $r(g_1, ..., g_m) \subseteq dom(g)$ та $dom(f) \subseteq \bigcap_{i=\overline{1,m}} dom(g_i)$. Та, якщо (g_1, g_2) є $\Diamond$-редукцією функції f, то

$$ran(f) \subseteq ran(g_1) \bigcup ran(g_2) \;\text{ і }\; dom(g_1) \bigcup dom(g_2) \subseteq dom(h).$$

Схеми циклювання визначаються схожим чином. Кортеж функцій $(g_1, ..., g_m)$, де $g_1, ..., g_m$ — функції виду $N^m \to N|_{m\in N}$, називається $*^{m+1}$-редукцією *m*-арної функції f, якщо $f = S^{m+1}(f, g_1, ..., g_m)$. Зв'язок $*^{m+1}$-редукції з операцією циклування описується через $h-m$-арний предикат та $g_1, ..., g_m, f$ — *m*-арні функції такі, що $f = *^{m+1}(h, g_1, ..., g_m)$. Тоді $(g_1, ..., g_m)$ — $*^{m+1}$-редукція функції f. Справедливість цього випливає з визначень сигнатурних операцій ППА.

Уведені редукції та їхні властивості визначають розв'язання задач шляхом покрокового розкриття структури генезису їхніх рішень. Це дозволяє отримувати коректні процедурні, алгоритмічні та програмні реалізації рішень, з їхніми нотаціями включно у мовах програмування. Таким чином, МПР забезпечує автоматичну побудову коректних реалізацій розв'язків задач. Кожен крок у цьому процесі відповідає актуалізації окремих оракулів генетичної структури розв'язання задачі оракульною схемою. Остання є логіко-математичною специфікацією рішення, обумовленого відповідним композитом. Технологічні системи програмування, які використовують такі специфікації, базуються на логіко-математичній імплементації ПР та дозволяють об'єктивізувати активну роль суб'єкта програмування.

Логіко-математичні релятивізації (ЛМР) прагматико-орієнтованих розв'язків задач можуть мати дуже різний рівень складності. Через це, для зручності класифікують ЛМР в залежності від типу задач, які можуть бути розв'язані за їхньою допомогою. Розв'язання задач вищих типів, таких як класи задач, задачі типу задач класів тощо. Основну роль у ЛМР задач вищих типів відіграють оракульні схеми [2]. Вони можуть бути дуже складними за своєю побудовою. Це пояснюється тим, що розв'язання задач вищих типів має інтеграційну складову, яку неможливо штучно обмежити заздалегідь. Проте, в [12], [13] показано, що для ЛМР в рамках ТСП можна обмежитися мікро-, макро-, біполярними оракульними схемами. Це дозволяє змістити акцент з ЛМР задач вищого типу на системи інтеграції розв'язання таких задач. Розглянемо деякі приклади такого підходу далі.

Розгляд логіко-математичних релятивізацій рішень задач в арифметичній ТСП на прикладах

Використання всюди істинного та всюди хибного предикатів p_T^1 та p_F^1 у арифметичній ТСП (АТСП) реалізує будь-які логічні зв'язки предикатів. Наприклад, предикати p_T та p_F можна визначити так: $p_T^1 = S^2\left(<, I_1^1, S^2\left(s^1, I_1^1\right)\right)$ і $p_F^1 = S^2\left(<, I_1^1, I_1^1\right)$. Тоді для будь-яких предикатів *p* та *q*:

$\neg\, p = \Diamond(p, p_F, p_T)$, $p \vee q = \Diamond\left(p, p_T^1, \Diamond\left(q, p_T^1, p_F^1\right)\right)$, $p \wedge q = \Diamond\left(p, \Diamond\left(q, p_T^1, p_F^1\right), p_F^1\right)$.

Мікрорівневу релятивізацію рішень тут, з огляду на досить обмежений набір базисних операцій, доповнено операцією усіченої різниці: $\dot{-}(x, y) = \begin{cases} x - y, \text{ якщо } y < x, \\ 0, \text{ інакше.} \end{cases}$

Розглянемо функцію $g^3 : N^3 \to N$ таку, що $g^3(\langle c,a,b\rangle) = \begin{cases} c+(a-b), \text{ якщо } b<a, \\ 0, \text{ інакше.} \end{cases}$ Особливість її в тому, що $g^3(\langle 0,a,b\rangle) = \begin{cases} a-b, \text{ якщо } b<a, \\ 0, \text{ інакше.} \end{cases}$ Звідси випливає, що кортеж функцій $\left\langle S^2\left(0^1, I_2^2\right), I_1^2, I_2^2, \right\rangle$ є S^4-редукцією функції $\dot{-}^2$. Адже, із зазначеного безпосередньо випливає, що для будь-якої пари $\langle a,b\rangle|_{a,b\in N}$ справедливо: $\dot{-}^2(\langle a,b\rangle) = S^4\left(g^3, S^2\left(0^1, I_2^2\right), I_1^2, I_2^2\right)(\langle a,b\rangle) = a \dot{-}^2 b$. Таким чином, терм $\dot{-}^2 = S^4\left(g^3, S^2\left(0^1, I_2^2\right), I_1^2, I_2^2\right)$ є логіко-математичною релятивізацією функції $\dot{-}^2$ з одним оракулом g^3. Для його актуалізації скористаємось такою властивістю функції $g^3 : g^3(\langle c,a,b\rangle) = g^3(\langle c+1,a,b+1\rangle)$. З цього випливає, що кортеж $< S^2\left(s, I_1^3\right), I_2^3, S^2\left(s, I_3^3\right) >$ є $*^4$-редукцією функції g^3. Таким чином

$$\dot{-}^2 = S^4\left(*^4\left(S^3\left(<^2, I_3^3, I_2^3\right), S^2\left(s, I_1^3\right), I_2^3, S^2\left(s, I_3^3\right)\right), S^2\left(0^1, I_2^2\right), I_1^2, I_2^2\right).$$

Отриманий розв'язок не містить оракулів і є логіко-математичною специфікацією класу задач, що містить лише функцію усіченої різниці $\dot{-}^2$. Тобто схемність структури цього розв'язку зведена тут до мікроінтеграції базисних функцій.

Макроінтеграційне середовище розглянемо на прикладі задачі обчислення значень функції, заданої скінченною сумою ряду:

$$F(x) = f\left(x, m(x), n(x)\right) = \sum_{i=m(x)}^{n(x)} g(x,i),$$

де $f(x,y,z)\big|_{x,y,z\in N}$ — функція підсумовування, тобто

$f(x,y,z) = \sum_{i=y}^{z} g(x,i)$, $m(x), n(x), g(x,y)\big|_{x,y,z\in N}$ — деякі арифметичні функції, причому для будь-якого $a \in N$ виконується $n(a) \geq m(a)$. Ці функції відіграють роль оракулів у розглядуваній постановці задачі і це основна її відмінність від попередньої.

Основою для розв'язання задачі є властивості функції f:

$$f\left(a, m(a), m(a)\right) = g\left(a, m(a)\right);$$

$$f\left(a, n(a), m(a)|_{n(a)>m(a)}\right) = f\left(a, n(a), m(a)-1\right) + g\left(a, m(a)\right).$$

Розглянемо функцію $q^4 = *^5\left(\neg\left(S^3\left(<^2, S^2\left(s, I_3^4\right), I_4^4\right)\right), S^3\left(+^2, I_1^4, g^2\left(I_2^4, I_4^4\right)\right), I_2^4, I_3^4, S^2\left(s, I_4^4\right)\right)$.

Її термальне позначення — $q(y,x,z,u)$. З вищенаведеної достатньої умови редукційності безпосередньо випливає, що $< S^3\left(+^2, I_1^4, g^2\left(I_2^4, I_4^4\right)\right), I_2^4, I_3^4, S^2\left(s, I_4^4\right) >$ є $*^5$-редукцією функції q^4.

Відповідно $q\left(r,a,n(a),m(a)\right)\big|_{r\in N} = r + f\left(a,n(a),m(a)\right) \equiv r + \sum_{i=m(a)}^{n(a)} g(a,i)$. Звідси випливає, що кортеж $< S^2\left(0^1, I_1^3\right), I_1^3, I_2^3, I_3^3 >$ є S^5-редукцією функції f^3 (у попередній (термальній) нотації $f(x,y,z)$). Тобто, $f^3 = S^5\left(q^4, S^2\left(0^1, I_1^3\right), I_1^3, I_2^3, I_3^3\right)$. Адже з визначення функцій $q(y,x,z,u), m(x)$ та $n(x)$ випливає, що $f\left(x, n(x), m(x)\right) = q\left(0, x, n(x), m(x)\right)\big|_{x\in N}$. Враховуючи унарність функції F^1 (термальна форма — $F(x)$) та її зв'язок з функцією f^3 остаточно маємо:

$$F^1 = S^4\left(f^3, I_1^1, S^2\left(n^1, I_1^1\right), S^2\left(m^1, I_1^1\right)\right).$$

Вибрана послідовність викладу спрямована на кращу ілюстрацію того, як побудувати оракульну схему для розв'язання задачі. Перехід до детальнішого представлення не є складним і буде показаний нижче.

На відміну від мікроінтеграційної релятивізації тут маємо вже макроінтеграційну релятивізацію — середовище, що підтримує вирішення класу задач підсумовування, основу якої складає схема взаємодії трьох оракулів – g^2, m^1 та n^1. Конкретні розв'язки задач з'являються у результаті актуалізації цих оракулів. Так, актуалізуючи функцію $g(x,i)$ як $g^2 = S^3\left(\times, I_1^2, I_2^2\right)$, а $m(x)$ та $n(x)$ як $m^1 = S^2\left(s, S^2\left(0^1, I_1^1\right)\right)$, $n^1 = S^2\left(s, I_1^1\right)$, отримуємо розв'язок для функції $\hat{F}(x) = \sum_{i=1}^{x+1} i \times x$ у вигляді мікроінтеграційної, вільної від оракулів, релятивізації функції $\hat{F}^1$, а саме:

$$\hat{q}^4 = *^5\left(\neg\left(s^1 I_3^4\right) <^2 I_4^4\right), I_1^4 +^2 I_2^4 \times I_4^4, I_2^4, I_3^4, s^1\left(I_4^4\right), \ \hat{f}^3 = S^5\left(\hat{q}^4, 0^1\left(I_1^3\right), I_1^3, I_2^3, I_3^3\right)$$

та остаточно $\hat{F}^1 = S^4\left(\hat{f}^3, I_1^1, s^1\left(I_1^1\right), s^1\left(0^1\left(I_1^1\right)\right)\right)$.

Отримані логіко-математичні релятивізації розв'язків задач в АТСП не просто демонструють розв'язки задач, але також точно відображають генезис цих розв'язків у вигляді складених термів, як результат застосування композитно-композиційних інтерфейсів АТСП. Коректність розв'язків випливає безпосередньо з їхньої побудови, і остаточне рішення в будь-якій системі нотацій зводиться до трансляції складеного терму в вибрану мову нотацій.

Синтаксичні особливості нотації результатів релятивізаційного підходу

Як вже зазначено, програмування включає породження та застосування композицій. Описане вище відображає лише породження композицій розв'язків задач. Композиційні терми розуміються як результати таких породжень. Вони мають свої переваги, але також мають серйозний недолік — вони є складними для сприйняття, оскільки відображають не розв'язки задач, а процес їхнього породження. Однак є можливість поліпшити цю ситуацію. Нотування не повинно відображати генезис рішення, а лише саме рішення, яке може бути обґрунтоване композиційним термом. Застосування породженої композиції допомагає спростити релятивізацію рішення завдяки інкапсуляції його генезисної складової. Особливо це стосується глибокої вкладеності композитів, яка є характерною для композиційних термів.

Ґрунтуючись на наведених визначеннях композитів АТСП, для отриманих композиційних термів матимемо такі дещо спрощені нотації розв'язків задач.

Для композиційного терму усіченої різниці здійснюючи покрокове «згортання» композитів, отримаємо $\dot{-}^2 = S^4\left(*^4\left(\left(I_3^3 <^2 I_2^3\right), s^1\left(I_1^3\right), I_2^3, s^1\left(I_3^3\right)\right), 0^1\left(I_2^2\right), I_1^2, I_2^2\right)$. Для композиційного терму функції $\hat{F}^1$, інтегруючи у єдиний терм вищенаведені вирази для $\hat{q}^4$ та $\hat{f}^3$ маємо:

$$\hat{F}^1 = S^4\left(S^5\left(*^5\left(\neg\left(s^1(I_3^4\right) <^2 I_4^4\right), I_1^4 +^2 I_2^4 \times I_4^4, I_2^4, I_3^4, s^1\left(I_4^4\right)\right), 0^1\left(I_1^3\right), I_1^3, I_2^3, I_3^3), I_1^1, n^1\left(I_1^1\right), m^1\left(I_1^1\right)\right).$$

Представлені розв'язки мають дві особливості: по-перше, вони побудовані відповідно до заздалегідь визначеної арифметичної ТСП, що гарантує їхню коректність; по-друге, їхня синтаксична форма може бути перетворена на відповідний семантичний терм у вибраній мові програмування за допомогою синтаксичної актуалізації композитних та функціональних структур з використанням відповідних оракулів. Цей процес може бути об'єктивізований за допомогою дефінітора відповідної мови програмування [8], [9].

Композито-композиційна релятивізація розв'язків задач має переваги перед конкретизованими специфікаціями у високорівневих мовах програмування. Використання такої форми специфікацій забезпечує прозорість переходів до нотацій у більшості високорівневих мов програмування.

Наведені приклади розв'язання задач за допомогою арифметичної ТСП відображають важливі загальні риси редукційного концептування оракульних схем. Такий підхід гарантує коректність

отримуваних розв'язків, що випливає безпосередньо з їхньої конструкції, а також дозволяє перейти від розв'язання окремих задач до розв'язання класів подібних задач. Отримані розв'язки можна реалізувати на різних синтаксичних програмних платформах.

Висновки

Розглянуто та обґрунтовано необхідність змін у технологічних підходах до програмування, а саме — перехід від індивідуального та суб'єктивного підходу до інтерактивнішого та технологізованого. Цей перехід дозволяє перетворити програмування з мистецтва на технологію. Для досягнення цієї мети розроблено концепцію відкрито-замкненого оракульного середовища програмування, яке базується на замкненій інтерактивній логіці та відкритому різноманітті продовжень, спрямованих на врахування активної ролі програміста.

Для забезпечення об'єктивізації активної ролі суб'єкта в контексті створення програм в технологічному середовищі, визначальним є концептування, що включає концепт, створений суб'єктом як певну специфікацію. Відкрито-замкнене концептопрограмне технологічне середовище, що розглядає програмування як суб'єктоорієнтовану продуктивну діяльність, базується на концептуванні.

Програмування розглянуто у контексті сучасного розуміння його як діяльності, яка є наслідком програмного уподібнення — взаємодії між активною та пасивною формами програми. Об'єктивізація цієї взаємодії є необхідною передумовою для осучаснення програмування як рефлексивно-транзитивного замикання, що породжується суб'єктом програмного уподібнення. Таким чином, розуміння взаємодії між активним та пасивним елементами програми є ключовим для розуміння програмування як цілісного процесу, що базується на програмному уподібненні.

Центральною і невід'ємною передумовою розуміння програмування як продуктивної діяльності є об'єктивізація його пасивної складової через програму, яка базується на концепті. В програмному уподібненні концептування забезпечує цілісність побудов, тоді як програма забезпечує їхню продуктивність. Цей процес становить основу відкрито-замкненого концептопрограмного технологічного середовища, яке реалізує розуміння програмування як продуктивної діяльності, обумовленої програмою та спрямованої на досягнення мети.

Композито-композиційні схеми забезпечують продуктивність програмування, де абстракції відповідних суб'єктних обумовлень розглядаються як оракули, а концептомонадне середовище програмування вважається оракульним середовищем. Інструмент логіко-математичних специфікацій семантико-синтаксичних аспектів програмування протестовано на задачах ППА, він продемонстрував загальні особливості використання технологічного середовища програмування для створення суб'єктно-орієнтованих технологічних систем програмування та їхнього використання.

Подальші дослідження у сфері технологізації програмування будуть фокусуватись на розширенні змістовних природничо-наукових досліджень і підтвердженні їхньої фактографії, а також розвитку відповідної фактології для оракульної схематизації концептування. Це є продуктивним методом релятивізації рішень для класів задач і розвиненого на її основі методу програмної релятивізації, який буде досліджуватись у майбутньому.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

[1] Д. Є. Кнут, *Искусство программирования. Основные алгоритмы*. М.: Вильямс, 2007.

[2] І. В. Редько, и П. О. Яганов, «Концептуальна модель технологічного середовища програмування,» *Наукові вісті КПІ*, № 1, с. 18-26, 2020. https://doi.org/10.20535/kpi-sn.2020.1.197953 .

[3] I. Redko, P. Yahanov, and M. Zylevich, "Concept-Monadic Model of Technological Environment of Programming," in *2020 IEEE 2nd International Conference on System Analysis & Intelligent Computing (SAIC)*, Kyiv, Ukraine, 2020, pp. 125-130. https://doi.org/10.1109/SAIC51296.2020.9239204 .

[4] І. В Редько, П. О. Яганов, і М. О. Зилевіч, «Редукційне концептування оракульних схем,» *Системні дослідження та інформаційні технології*, № 1, с. 21-33, 2021. https://doi.org/10.20535/SRIT.2308-8893.2021.1.02 .

[5] И. В. Редько, «Прагматические основания дескриптивных сред,» *Проблемы программирования*, № 3, с. 3-25, 2005.

[6] Г. В. Лейбниц, *Сочинения в 4-х томах*. М.: Мысль, 1982.

[7] В. Н. Редько, «Композиции программ и композиционное программирование,» *Программирование*, № 5, с. 3-24, 1978.

[8] В. Н. Редько, «Дефиниторы и метод дефиниторного процессирования,» *Кибернетика*, № 6, с. 52-56, 1974.

[9] И. А. Басараб, Н. С. Никитченко, и В. Н. Редько, *Композиционные базы данных*. К.: Лыбидь, 1992.

[10] В. Н. Редько, «Основы програмологии,» *Кибернетика и систем. анализ*, № 1, с. 35-57, 2000.

[11] В. Н. Редько, «Основы композиционного программирования,» *Программирование*, № 3, с. 3-13, 1979.

[12] В. Н. Редько, Н. В. Гришко, и И. В. Редько, «Экспликативное программирование в среде логико-математических спецификаций,» *УкрПРОГ'98,* с.71-76, 1998.

[13] И. В. Редько, и Н. В. Гришко, «Экспликативное программирование в среде интеграции,» *Проблемы программирования*, № 2, с. 59-65, 2004.

[14] D. I. Redko, I. V. Redko, P. O. Yahanov, and T. L. Zakharchenko, “Compositional basis in programmer activity,” *Системні дослідження та інформаційні технології*, № 4, с. 83-96, 2015.

[15] А. И. Мальцев, *Алгоритмы и рекурсивные функции*. М.: Наука, 1965.

[16] Д. Б. Буй, *Примитивные программные алгебры*. Киев, 1984.

[17] А. Пуанкаре, *О науке*. М.: Наука, 1990.

Рекомендована кафедрою програмного забезпечення ВНТУ

Стаття надійшла до редакції 14.03.2023

Редько Ігор Володимирович — д-р фіз.-мат. наук, професор кафедри конструювання електронно-обчислювальної апаратури, e-mail: redkoigor@kpi.ua ;

Зилевіч Максим Олегович — аспірант кафедри конструювання електронно-обчислювальної апаратури, e-mail: m.zylevich@kpi.ua .

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ

I. V. Redko[1]
M. O. Zylevich[1]

Theoretical Foundations of Software Relativization in Technological Programming Systems

[1]National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”

The research claims the need for paradigmatic changes in the technological foundations of programming, which involve a transition from an individual-subjective programming paradigm to an intersubjective one. The proposed intersubjective paradigm is based on the understanding of programming as an activity conditioned by program likeness, and subject-oriented complementarity of its active and passive forms. To implement such a transition, the objectification of active-passive complementarity is considered, which is the main prerequisite for the real modernization of the understanding of programming as a reflexive-transitive closure of the programming analogy generated by the subject. The Determinant for the objectification of the active role of the subject in the context of technological programming environments is a concept — essence—simile, presented in the form of one or another specification. Passive constituent conditioning is objectified by similitude as a whole consequence of assimilation, namely by a program. To achieve this goal, a tool for logical-mathematical specifications of semantic-syntactic aspects of programming is proposed. This tool was tested on examples that demonstrated the general features of the application of the technological programming environment to the generation of subject-oriented technological programming systems and their use. In particular, the use of this tool allows to create more efficient technological programming systems that have higher quality and flexibility in performing tasks. The obtained results confirm the need for paradigmatic changes in the technological foundations of programming and are one of the possible options for solving the problem of paradigmatic changes and further technologization of programming, and also demonstrate the general features of the application of a technological programming environment for the generation of subject-oriented technological programming systems and their use.

Keywords: concept, monad, composite, composition, reduction, programming environment, oracle schemes.

Redko Ihor V. — Dr. Sc. (Phys.-Math.), Professor of the Chair of Electronic Computing Equipment, e-mail: redkoigor@kpi.ua ;

Zylevich Maksym O. — Post-Graduate Student of the Chair of Electronic Computing Equipment, e-mail: m.zylevich@kpi.ua