УДК 681.3.06
DOI: 10.20535/SRIT.2308-8893.2021.1.02

РЕДУКЦІЙНЕ КОНЦЕПТУВАННЯ ОРАКУЛЬНИХ СХЕМ

І.В. РЕДЬКО, П.О. ЯГАНОВ, М.О. ЗИЛЕВІЧ

Анотація. Роботу спрямовано на розвиток інтерсуб'єктивної парадигми та активної ролі суб'єкта у редукційному концептуванні. Для цього загальну оракульну схему концептування конкретизовано взаємодоповненням композиції і декомпозиції як експлікацій синтезу та аналізу сутностей. Прагматико-обумовлене збагачення цього взаємодоповнення здійснено із залученням композиційного програмування та іменних моделей даних, функцій і композицій. Розглянуто оракульну схему редукції, сенс якої полягає в тому, що вона, спираючись на наявні композиції, природним чином імплементує парадигму «поділяй і володарюй» у розумінні активної ролі суб'єкта в концептуванні, підтримуючи реальне взаємодоповнення декомпозиційного і композитного методів концептування. Наведено репрезентативні приклади редукційного концептування, які обґрунтовують технологію вирішення програмістських завдань.

Ключові слова: програмування, інтерсуб'єктивна парадигма, оракульна схема, концептування, редукція.

ВСТУП

Зародження, розвиток, становлення, стагнація та занепад будь-яких технічних систем і пов'язаних з ними технологічних процесів відбуваються за об'єктивними законами діалектики, проходячи ряд характерних етапів від бурхливого зростання до системної кризи, яка спонукає шукати вихід із ситуації, що склалася. Часто результативним вирішенням є зміна парадигми та створення і розвиток нової концепції, завдяки якій система у своєму зростанні виходить на новий рівень. Програмування як вид інформатико-технологічної діяльності не є винятком. Сьогодні стрімкий розвиток програмістської творчості, попри беззаперечні успіхи, нагромадив чимало проблем, ігнорування яких або відсилання їх на людський фактор веде до «бігу по колу» і відтворення помилок, що дедалі більше поглиблює кризу.

Розглянемо зміст програмування, яке зазвичай сприймають як діяльність з отримання результату — готової програми, тобто як процес реалізації плану (програми) досягнення цього результату. У такому разі варто зазначити, що програмування як діяльність провадиться в певних умовах, які формуються у невідривному зв'язку з програмою. Можна стверджувати про наявність обумовленості між програмуванням і програмою. Цей причинно-наслідковий зв'язок є суб'єктно-об'єктним, у якому суб'єкт — програміст, будучи вагомим зовнішнім чинником, реалізовує свій задум, спираючись на власні уявлення про способи, методи, евристики, знахідки і запозичення досягнень результату в межах об'єкта — процесу створення програми. Об'єктивізм процесу виражається у можливості схематизації програмування генетичними структурами з властивими їм структурними сутностями. Генезис визначає структуру програми, її внутрішню логічну цілісність, якій підпорядковується результат програмування і нею обумовлюється.

У такому причинно-наслідковому зв'язку місце і роль суб'єктно-об'єктної взаємодії для сучасного розуміння програмування є ключовими і мають розглядатись не поза програмістською діяльністю, як це зазвичай відбувається, а безпосередньо як самостійний об'єкт дослідження. З одного боку, суб'єкт програмування не обмежений ніякими умовами у процесі програмотворення, крім головної, яку висловлює замовник, — програма повинна забезпечувати очікуваний результат. При цьому програміст «поділяє і володарює», обираючи на власний розсуд методи і засоби досягнення результату, використовуючи власний арсенал прийомів, підтверджуючи своїми діями поняття «мистецтво програмування». Його активна роль полягає не у залученні до схематизації об'єктивного причинно-наслідкового зв'язку між результатом і процесом його отримання у вигляді конкретних схем, а у формуванні власної авторської «функції множини дій», за допомогою якої він вирішує поставлене завдання без належної конкретизації проміжних результатів. Про технологію програмування у цьому випадку говорити важко, оскільки досягнення кожного суб'єкта програмування реально невіддільні від нього, бо не підтримують взаємодоповнення фактографії і фактології інформатико-технологічної діяльності. Загальновідомим у програмістському середовищі є твердження про те, що невдалу програму простіше створити заново, ніж виправити.

Активна роль суб'єкта повинна виразитись у конкретизації діяльності обумовлення, проваджуваної за допомогою плану програмування, актуалізацією значущих для процесу якісних (інтенсіональних) суб'єктивних обумовлень. Таким чином, традиційна індивідно-суб'єктивна парадигма, у межах якої програма розглядається як результат програмування, має змінитись інтерсуб'єктивною парадигмою, згідно з якою програмування — це діяльність суб'єкта зі створення програми (плану) цієї діяльності.

Суб'єкт програмування здійснює обумовлення в активній і пасивній формах. Активна форма — це безпосереднє накладання умови носієм таких умов (суб'єктом обумовлення) на програмування залежно від того, яким арсеналом засобів він оперує. Програміст впливає на результат і на спосіб досягнення цього результату — програмування як процесу. Активна форма виявляється безпосередньо в прагматиці використання для досягнення мети активних видів обумовлень, які створені не тільки в межах вирішення конкретного завдання, але і залучені, зокрема, з інших методів, що розвиваються на основі традиційного математичного апарату для нотації результату, а також денотативних прийомів.

Пасивна форма зумовлена наслідком активізації діяльності суб'єкта і опосередковано впливає на наслідок планування програми. Цей вплив виражається у залученні до процесу вирішення завдань, які вже відбулись, — або в межах цього процесу, або поза ним. Ці вирішення матеріалізовані у результатах діяльності суб'єкта і його власного розуміння такої діяльності. Зазвичай із цим пов'язують досвід, здобутий у процесі провадження активної діяльності та активного обумовлення цієї діяльності. В інтерсуб'єктивній парадигмі активно-пасивне обумовлення збагачує поняття програми як сутності, оскільки визначає роль суб'єкта не поза межами процесу програмування, а безпосередньо у ньому, знаходячи своє відображення, зокрема, у формі семантичного терму, фіксуючи ключові моменти розуміння діяльності, що спонукала до появи результату.

Мета роботи — подальше розвинення інтерсуб'єктивної парадигми та об'єктивізації активної ролі суб'єкта у редукційному концептуванні оракульних схем.

Для досягнення мети роботи загальну оракульну схему концептування конкретизовано взаємодоповненням композиції і декомпозиції як аналогів синтезу та аналізу сутностей, а прагматико-обумовлене збагачення цього взаємодоповнення розглянуто на репрезентативних прикладах редукційного концептування із залученням композиційного програмування та іменних моделей даних, функцій і операцій.

ОРАКУЛЬНА СХЕМАТИЗАЦІЯ КОНЦЕПТУВАННЯ

У працях [1–3] обґрунтовано засадниче для інтерсуб'єктивної парадигми програмування положення, що поза цілісним розумінням програмування немає його продуктивного розуміння. Концептуванню відведено роль носія можливих збагачень цього змістовного положення. Неможливість об'єктивного зведення один до одного цих видів розумінь зумовила відкритозамкненість та суб'єктно-об'єктність схеми концептування [3].

Згадані у цих працях схеми є концептами для різних композитних і композиційних концептувань. Останні отримуються за рахунок актуалізації оракулів [4, 5], що входять до схем. Із самої побудови схем випливає, що вони продукують функціональні сутесутнісні залежності вигляду $Ent \overset{Comt}{\to} CtMon$ або $Ent \overset{Comp}{\to} CMon$, де $CtMon$ — збагачення оракула Mon за рахунок актуалізації концепту композитом; $Comt$ — оракул «композит»; $Comp$ — оракул «композиція»; $CMon$ — збагачення оракула Mon за рахунок актуалізації концепту композицією [4].

Отже, і композитні, і композиційні монади можуть бути експліковані як композиції сутностей: $CtMon \Rightarrow Comt(Ent_{i_1},....,Ent_{i_k})$ і $CMon \Rightarrow$ $\Rightarrow Comp(Ent_{j1},....,Ent_{j_p})$, де $Ent_{i_1},....,Ent_{i_k}, Ent_{j_1},....,Ent_{j_k}$ — деякі сутності, а $\Rightarrow$ означає «експлікативно зводиться». Наявні у схемі концептування оракули є «реперними точками» реалізації активної ролі суб'єкта в концептуванні. Виходячи з того, що ключовою ланкою реалізації активності суб'єкта є композит як концепт активності суб'єкта і його зв'язок з композицією, можна констатувати, що активність суб'єкта є концептуванням, концептом якого є композит, — базова для суб'єкта генна структура.

Розглядаючи ці експлікативні зведення в контексті концептування і проектуючи їх на відповідні функціональні залежності, отримуємо такі оракульні схеми: $Ent \overset{Comt}{\to} Comt(Ent_{i_1},....,Ent_{i_k})$ і $Ent \overset{Comp}{\to} Comp(Ent_{j_1},...,Ent_{j_p})$. Особливість кожної з них полягає у тому, що перша використовує базову композицію, але при цьому на сутності $Ent_{i_1},....,Ent_{i_k}$ ніяких додаткових обмежень не накладається. У другій же, навпаки, композиція не обов'язково базова, швидше вона похідна від покроковості застосування композитів, зате сутності $Ent_{j_1},....,Ent_{j_k}$ є елементарними, тобто достатньо деталізованими для суб'єкта. Це створює можливість передбачення наслідку концептування як композиції елементарних, для суб'єкта, сутностей.

У наведених схемах явно відображений факт взаємодоповнення двох методів дослідження концептування — синтезу та аналізу, композиції і декомпозиції. Синтез поданий в них оракулами $Comt$ і $Comp$, а аналіз (декомпозиція) — $Ent_{i_1},....,Ent_{i_k}, Ent_{j_1},....,Ent_{j_k}$.

Звернемо увагу на те, що перша схема оперує базовими для суб'єкта композиціями (композитами). Це дозволяє здійснити ще один крок до продуктивного збагачення розуміння програмування. Будемо говорити про те, що кортеж $< Ent_{i_1},...., Ent_{i_k} >$ є $Comt$ -редукцією сутності Ent , якщо існує функціональна залежність $Ent \overset{Comt}{\rightarrow} Comt(Ent_{i_1},...., Ent_{i_k})$. З наведеного розгляду безпосередньо випливає твердження про оракульну схематизацію редукції.

Твердження. Кортеж $< Ent_{i_1},...., Ent_{i_k} >$ є $Comt$-редукцією сутності Ent, якщо справедлива $Ent = Comt(E_{i_1},...., E_{i_k})$.

Змістовний сенс наведеної оракульної схеми полягає у тому, що вона природним чином імплементує парадигму «поділяй і володарюй» в розумінні активної ролі суб'єкта в концептуванні, підтримуючи реальне взаємодоповнення декомпозиційного і композитного методів концептування [3]. Таким чином, композит $Comt$ як актуально задана базова композиція є концептом Ent, тобто Ent є монадою. Цю особливість буде використано у наведених нижче прикладах редукційного концептування.

РЕПРЕЗЕНТАТИВНІ ПРИКЛАДИ РЕДУКЦІЙНОГО КОНЦЕПТУВАННЯ

Природа композитів і композицій в цілому релятивна, що істотно залежить як від суб'єкта, так і від розглядуваних сутностей. У свою чергу, розуміння сутності залежить від композитів, що залучаються у розгляд. Тому реальне проникнення в інтерсуб'єктивну природу композитів і композицій можливе тільки у взаємодії з прагматико-обумовленим збагаченням цього взаємодоповнення. У цьому напрямі отримано велику кількість результатів різної глибини і значущості [6–11]. Детальний розгляд такої фактографії виходить за межі цієї роботи. Тому доцільно поповнити її якісно іншими простими і, разом з тим, репрезентативними прикладами редукційного концептування.

Розв'язання будь-якої задачі, як відомо, є інтеграцією розв'язань її підзадач. Якщо задача проста, то інтеграція тривіальна і, як правило, явно не виділяється. У разі ж коли задача складна, інтеграційний аспект її розв'язання домінує, адже власне саме ним і визначається складність.

Відомо, що рівень складності розв'язання реальних задач визначається, головним чином, складністю їх інтеграційних складових. Зважаючи на це, подальші побудови проведемо «від простого до складного», розглянувши на прикладах нескладних задач числового аналізу їх розв'язання в середовищах мікро- та макроінтеграції. Як платформу розгляду використаємо композиційне програмування та іменну модель даних, функцій і операцій, а як композити — операції мультиплікування $\circ$, розгалуження IF , циклування WD і найпростіші похідні від них композиції, що уточнюють найбільш вживані та прості способи генезису одних програм з інших [6, 7]. Надалі під даними, функціями та операціями, якщо не зазначено інше, розумітимемо іменні дані, іменні функції та іменні операції відповідно.

Проведене змістовне розгортання і роз'яснення концептування достатньо обгрунтовує точку зору на нього як на покроковість редукувань, що зводиться до пошуку підхожої $Comt$-редукції для індукованого відповідною функціональною залежністю рівняння $Ent = Comt(E_{i_1},...., E_{i_k})$. Під розв'язком тут розуміється кортеж $< Ent_{i_1},...., Ent_{i_k} >$ такий, що справедли-

вою є тотожність $Ent \equiv Comt(Ent_{i_1},....,Ent_{i_k})$. Запис $Comt(Ent_{i_1},....,Ent_{i_k})$ означає застосування k-арного композита $Comt$ до кортежу $< Ent_{i_1},... ...,Ent_{i_k} >$, тобто $Comt(Ent_{i_1},....,Ent_{i_k}) \stackrel{\text{def}}{=} Ap(Comt,< Ent_{i_1},....,Ent_{i_k} >)$, де Ap розуміється традиційно як аплікація [9], а $\stackrel{\text{def}}{=}$ як рівність за визначенням.

Розглянемо застосування апарату простої WD-редукції, рухаючись від простого до складного. Така редукція є кортежем $< g, p >$, що є розв'язком рівняння $f = WD(g, p)$, де g, p — деякі функція та предикат. Для пошуку розв'язку корисними є наслідки згаданого твердження про схематизацію редукції, які продуктивно збагачують його, при цьому істотно звужуючи коло пошуку редукції.

Наслідок 1. $< g, p >$ є WD-редукцією функції f, якщо $f = WD(p, g)$.

Наслідок 2. Щоб кортеж $< g, p >$ був WD-редукцією функції f, необхідно, щоб виконувалась рівність $g \circ f = f$, де $\circ$ — операція мультиплікування.

Проілюструємо це на конкретних прикладах. Для цього, передусім, звернемося до найпростішого класу задач числового аналізу, що складається з однієї задачі — обчислення $\sqrt{x}$ із заданою точністю ε, де x і ε — додатні дійсні числа.

Відомо, що послідовність $y_0, y_1, y_2, \ldots$, у якій $y_0 = a, y_{i+1} = \frac{1}{2}\left(y_i + \frac{x}{y_i}\right)|_{i=0,1,2,\ldots}$, де a — деяке додатне дійсне число, незалежно від a, збігається до $\sqrt{x}$. Звідси випливає, що процес обчислення $\sqrt{x}$ із заданою точністю може бути зведений до деталізації іменної функції f [7, 11], яка іменній множині $\{(u, x), (v, \varepsilon), (w, a)\}$ ставить у відповідність іменну множину $\{(w, y_n)\}$, де y_n — перший член зазначеної послідовності, для якого виконується умова $| y_n^2 - y_{n-1}^2 | < \varepsilon$.

Знайдемо WD-редукцію функції $f_{\sqrt{x}}$. Розглянемо наступні іменну функцію $g : \left\{(u, x), (v, \varepsilon), (w, a)\} \to \{(u, x), (v, \varepsilon), (w_{pr}, a), \left(w, \frac{1}{2}\left(a + \frac{x}{a}\right)\right)\right\}$ та іменний предикат $p : \{(v, \varepsilon), (w, a), (w_{pr}, b)\} \to \begin{cases} \text{True}, & | a^2 - b^2 | \geq \varepsilon, \\ \text{False}, & | a^2 - b^2 | < \varepsilon. \end{cases}$ Легко переконатися у справедливості рівності $f_{\sqrt{x}} = g \circ f_{\sqrt{x}}$ і, як наслідок, у тому, що $< g, p >$ дійсно є WD-редукцією функції. Адже дійсно, якщо $| a^2 - \left(\frac{1}{2}\left(a + \frac{x}{a}\right)\right)^2 | < \varepsilon$, то $f_{\sqrt{x}}\left(\left\{(u, x), (v, \varepsilon), \left(w, \frac{1}{2}\left(a + \frac{x}{a}\right)\right)\right\}\right) = \{(w, y_n)\}$. Отже, можна зробити висновок: $f_{\sqrt{x}} \equiv WD(g, p)$. При цьому правильність висновку безпосередньо випливає з побудови. Особливістю цього вирішення є те, що його синтаксичне оформлення зводиться до трансляції відповідного семантичного терму в обрану мову програмування.

```
Procedure G(x:double; var rz, zr:double);
        begin
                zr:=rz;
                rz:=0,5*(rz+x/rz);
        end;
```

Рис. 1. Приклад процедури *G*

Продемонструємо це на прикладі мови програмування Pascal. Почнемо із функції редукції g та предиката p. Процедура *G*, очевидно, подає функцію g у синтаксисі мови Pascal (рис. 1).

Процедура-функція P у тому ж синтаксисі подає предикат p (рис. 2).

Насамкінець подамо шукану функцію $f:\{(u,x),(v,\varepsilon),(w,a)\}\to\{(w,y_n)\}$ у Pascal-синтаксисі (з деякими непринциповими скороченнями, зробленими заради компактності викладення матеріалу, рис. 3).

```
Function P(rz,zr: double):boolean;
        begin
                if abs(sqr(rz)-sqr(zr))≥v then
                        P:=True
        else
                        P:=False;
        end;
```

Рис. 2. Приклад процедури-функції *P*

Зазначимо, що синтаксичне подання шуканого розв'язку може породжуватись в автоматичному режимі за допомогою спеціалізованого програмного забезпечення — дефінітора мови Pascal [3, 7]. Таким чином, необтяженість семантичної специфікації розв'язання задачі не тільки не заважає отриманню його синтаксичного подання, а є суттєвою перевагою такої специфікації перед надто конкретизованими високо рівневими мовами програмування.

```
Program F;
        var u,v,w,wpr: double;
        Procedure G(x:double; var rz,zr:double);
        ...
        Function P(rz,zr:double):boolean;
        begin
        ...             // Блок уведення значень u,v,w

        if (u>0) and (v>0) and (w>0) then

                        begin
                        G(u, w, wpr);
                        while P(w, wpr) do G(u, w, wpr);
                        writeln(w);
                        end;
        end.
```

Рис. 3. Приклад Pascal-подання шуканої функції

Даний приклад демонструє задачу, що розв'язується «в один крок». Для розв'язань таких задач не потрібно залучати оракульні схеми концептування, тому і інтеграційна складова тут є тривіальною. Наступний приклад вже на макрорівні ілюструє інтеграційний аспект концептування і значно виразніше демонструє переваги редукційного концептування. Розглянемо клас спеціальних рівнянь вигляду $x=\varphi(x)$, де функція $\varphi(x)|_{x\in R}$ задовольняє такі дві умови:

1) вона визначена і неперервно диференційовна на всій числовій прямій;

2) існує таке дійсне число $p<1$, що для всіх x справедлива нерівність $|\varphi'(x)|\le p$.

Відомо, що стосовно такого класу рівнянь метод простих ітерацій збігається. Причому розв'язком є границя послідовності $\{x_i\}_{i=0,1,2,..}$, де x_0 — будь-яке дійсне число, а $x_{i+1}=\varphi(x_i),\ i=0,1,2,...$

Із наведених умов випливає, що концептування задачі пошуку розв'язку рівнянь вигляду $x=\varphi(x)$ можна звести до деталізації функції

$f:\{(v,x_0),(u,\varepsilon)\}\to\{(v,x_n)\}$, де x_n — перший член послідовності наближень, для якого виконується умова $|x_{n-1}-x_n|<\varepsilon$. Очевидно, що кортеж $<g,p>$, де $g:\{(v_{pr},a),(v,b)\}\to\{(v_{pr},b),(v,\varphi(b))\}$, а $p:\{(v_{pr},a),(v,b),(u,\varepsilon)\}\to$ $\to\begin{cases}\text{True}, |a-b|\geq\varepsilon\\ \text{False}, |a-b|<\varepsilon\end{cases}$ є розв'язком рівняння $f=WD(E_1,E_2)$. Тобто $f\equiv WD(g,p)$.

Особливість отриманого розв'язку полягає в тому, що він є вже оракульною схемою розв'язання класу задач, інакше кажучи, є схемою монади. Оракулом тут є $\varphi:\{(v,b)\}\to\{(v,\varphi(b))\}$ — іменна специфікація $\varphi(x)|_{x\in R}$. Схема перетворюється в конкретну монаду після заміни φ у схемі конкретною функцією, що задовольняє наведені вище дві умови. Такими, наприклад, є функції $\varphi:\{(v,b)\}\to\left\{\left(v,\frac{\cos(b)}{2}\right)\right\}$, $\varphi:\{(v,b)\}\to\left\{\left(v,\frac{\sin(b)+\cos(b)}{3}\right)\right\}$ і т.ін. У цій схемі оракульну взаємодію подано рудиментарно, оскільки в ній усього один оракул — φ. При цьому неможливість коректного Pascal-подання отриманої схеми через потенційну нескінченість класу описуваних нею окремих розв'язків не заважає використовувати її для породження Pascal-специфікацій конкретних монад. Для цього необхідно лише монадизувати оракул φ.

Нехай $\varphi(x)=\sum_{i=1}^{\infty}(-1)^i\frac{x^{2i}}{2(2i)!}$, $|x|<\infty$, що задовольняє наведені вище умови, і Pascal-платформа реалізації оракула φ містить, крім стандартних математичних функцій мови Pascal, і функцію факторіала FC.

Тоді для будь-якого дійсного a послідовність $y_0,y_1,y_2,\ldots$, де $y_0=0$, $y_{k+1}=\sum_{i=1}^{k+1}(-1)^i\frac{a^{2i}}{2(2i)!}=y_k+(-1)^{k+1}\frac{a^{2(k+1)}}{2(2(k+1))!}|_{k=0,1,2}$, збігається до $\varphi(a)$. Для обчислення $\varphi(a)$ із заданою точністю до дійсного додатного числа ε корисною є функція q, що іменній множині $\{(v,a),(t,\varepsilon),(w,b)\}$ ставить у відповідність іменну множину $\{(w,y_n)\}$, де y_n — перший член зазначеної послідовності, для якого виконується умова $\left|\frac{a^{2n}}{2(2n)!}\right|<\varepsilon$. Очевидно, що $\Phi=q_1\circ q$, де $q_1:\{(w,b),(s,m)\}\to\{(w,0),(s,0)\}$, а $\Phi:\{(v,a),(t,\varepsilon)\}\to$ $\to\{(w,y_n)\}$ — іменна функція, яка обчислює наближене значення $\varphi(a)$ із заздалегідь заданою точністю. Тобто пара $<q_1,q>$ є $\circ$-редукцією функції Φ. Функція q_1 обнулює «комірки» w і s, тобто подається операторами присвоювання $w:=0;$ $s:=0;$ і тому не потребує подальшої деталізації. Що стосується функції q, то аналогічно до попереднього, пара $<r,p>$, де $r:\{(v,a),(w,b),(s,k)\}\to\left\{(w,b+(-1)^{k+1}\frac{a^{2(k+1)}}{(2(k+1))!}),(s,k+1)\right\}\Big|_{a,b\in R,k\in N}$ та

$$p:\{(t,\varepsilon),(v,a),(s,k)\}\to\begin{cases}\text{True}, & \left|\dfrac{a^{2n}}{2(2n)!}\right|\geq\varepsilon,\\ \text{False}, & \left|\dfrac{a^{2n}}{2(2n)!}\right|<\varepsilon\end{cases}$$ є *WD*-редукцією функції q. І, таким чином, $\Phi = q_1 \circ WD(r,p)$.

Відповідне подання функції *r* здійснюється Pascal-процедурою (рис. 4).

```
Procedure R(var v, w: double; var s: integer);
        begin
          w:=w+(((exp((s+1)*ln(-1))*(exp(2*(s+1)*ln(v))))/(2*FC((2*(s+1)))));
          s:=s+1;
        end;
```

Рис.4. Приклад процедури *R*

Предикат p подається процедурою, аналогічною до наведеної у попередньому прикладі процедури-функції *P* (рис. 5).

```
Function P(v, t: double; s:integer):boolean;
begin
        if s>0 then
        begin
        if abs((((exp((s+1)*ln(-1))*(exp(2*(s+1)*ln(v))))/(2*FC((2*(s+1))))))>=t
                        then
                                P:=True
                        else
                                P:=False;
        end;
end;
```

Рис. 5. Приклад процедури-функції *P*

Тепер нескладно надати Pascal-подання для функції Φ (рис. 6).

```
Function FI(v,t:double):real;
Var
i:integer;
w:double;
begin
 w:=0; i:=0;
   while P(abs((((exp((i+1)*ln(-1))*(exp(2*(i+1)*ln(v))))/(2*FC((2*(i+1)))))))>t
   do
     begin
       R(v,w,i);
     end;
 F:=w;
end;
```

Рис. 6. Приклад процедури-функції *FI*

Продовжуючи побудови, приходимо до Pascal-програми (рис. 7).

Розглянуті приклади демонструють важливі особливості редукційного концептування оракульних схем. Це, по-перше, можливість переходу від розв'язань окремих задач до розв'язань класів подібних задач з гарантова-

ною коректністю отримуваних редукційним концептуванням розв'язків за їх побудовою. І, по-друге, отримані концептуванням розв'язки можуть бути реалізовані на різних синтаксичних програмних платформах.

```
Program F;
    var u,w,wpr,t:double;
       s:integer;
    Function G(e:double; x:double):double;
    begin
                  x:=FI(x,e); G:=x;
    end;
    Function P(x, v:double):boolean;
    begin
                  if abs(x- G(x))>=v then P:=True else P:=False;
    end;
    Procedure R(x:double; var rz,zr:double; var s:integer);
    begin
                ...
    end;
    Function FI(v,t:double):real;
    begin
                ...
    end;
    // Блок уведення значень u,t,w
    ...
    begin
      if (t>0) then
                              begin
                              w:=0;
                              while P(w, t) do G(w, t);
                              readln(w);
                              end;
    end.
```

Рис. 7. Приклад програми *F*

Прагнення до першого було і залишається спонукальним мотивом створення більшості парадигм програмування. У тому чи іншому вигляді важливість цього знайшла своє відображення, зокрема, у тьюрінговських лекціях Джона Бекуса, Едсгера Дейктра, Джона Мак-Карті, Роберта Флойда [12]. Видатні вчені, які зробили вагомий внемок у розбудову засад, у тому числі і комп'ютерної науки та програмування у цілому, такі як Алонзо Чорч, Стівен Кліні, Гаскелл Каррі, Ніклаус Вірт, Алан Кей та їх послідовники у своїй діяльності також керувались і керуються проблемами підвищення продуктивності та забезпечення коректності отримуваних розв'язків [13–26]. Реінжиніринг програмного забезпечення з лавиноподібним зростанням інвестицій у його розроблення став самоочевидним трендом розвитку інформаційних технологій та програмування [27]. Віддаючи належне визначним досягненням, зробленим у цьому напрямі, тим не менше, необхідно звернути увагу на те, що всі вони з об'єктивних причин слабко інтегровані між собою. Адже їх адекватна інтеграція потребувала розгляду ряду питань, для вирішення яких не було достатньої фактографії. Такими, зокрема, є питання об'єктивізації суб'єктивних впливів на розв'язання задач переходу від систем, орієнтованих на замкнуті у конкретиці розв'язки, до систем, орієнтованих на розв'язання класів подібних задач.

Що ж до другого, то спрямованість редукційного концептування на притаманні програмуванню генетичні структури дозволяє враховувати активну роль суб'єкта у ньому, на відміну від більшості традиційних підходів, які орієнтовані на синтаксичну нотацію результатів свідомого чи несвідомого застосування таких генетичних структур. Таким чином, маємо достатньо підстав розглядати редукційне концептування оракульних схем як продуктивну експлікацію програмування.

ВИСНОВКИ

У роботі обгрунтовано ключові поняття інтерсуб'єктивної парадигми програмування із залученням оракульних схем концептування. Розглянуто експлікативні зведення концептування до нових оракульних схем, які є концептами для різних композитних і композиційних концептувань. Інструментом такого зведення є конкретизація оракулів, що входять до цих схем. Продуктивне збагачення розуміння програмування знайшло своє відображення у твердженні про оракульну схематизацію редукції.

Розвинуто і збагачено новим змістом програмування як діяльність, орієнтовану не тільки на нотацію одержуваних розв'язків, але таку, яка здатна забезпечувати продуктивну діяльність суб'єкта для їх отримання із залученням розвинених засобів програмування, адаптованих для реалізації активної ролі суб'єкта. Для цього необхідно вирішувати проблему побудови програмологічних засад продуктивної діяльності і розвитку адаптивних інформатико-технологічних середовищ.

Наведено репрезентативні приклади редукційного концептування, які обгрунтовують оракульну схематизацію як технологію вирішень програмістських завдань.

Подальші дослідження з цієї проблематики будуть спрямовані на розширення змістовних природничо-наукових досліджень, фактографії, що підтверджує їх, і розвиток відповідної фактології для оракульної схематизації концептування як продуктивного засобу специфікації розв'язків класів задач і розвинених на її основі редукційних методів програмування.

ЛІТЕРАТУРА

1. І.В. Редько та П.О. Яганов, "Концептуальна модель технологічного середовища програмування", *Наукові вісті КПІ*, № 1, с. 18–26, 2020. doi: 10.20535/kpi-sn.2020.1.197953.
2. I. Redko, P. Yahanov, and M. Zylevich, "Concept-Monadic Model of Technological Environment of Programming", *2020 IEEE 2nd International Conference on System Analysis & Intelligent Computing (SAIC)*, Kyiv, Ukraine, 2020, pp. 125–130. doi: 10.1109/SAIC51296.2020.9239204.
3. І.В. Редько, Д.І. Редько, та Т.Л. Захарченко, *Концептологічні основи проектування*. Київ: Компринт, 2016.
4. В.Р. Карымов, "Арифметическая и гиперарифметическая вычислимость относительно вычислений с ограничениями", *Математика и механика*, № 5, с. 48–52, 2010.
5. M. Trakhtenbrot, "Mutation Patterns for Temporal Requirements of Reactive Systems", *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Tokyo, 2017, pp. 116–121. doi: 10.1109/ICSTW.2017.27
6. А.И. Мальцев, *Алгоритмы и рекурсивные функции*. Москва: Наука, 1965.

7. И.А. Басараб, Н.С. Никитченко, и В.Н. Редько, *Композиционные базы данных*. Киев: Лыбидь, 1992.
8. Д.Б. Буй, Е.В. Шишацкая, S. Fabunmi, K. Mohammed, "Математичні основани алгоритмів лінеаризації: рефлексивно-транзитивне замикания бінарних відношень", *Вісник Харківського національного університету імені В.Н. Каразіна*, № 29, с. 19–33, 2016.
9. Х. Барендрегт, *Лямбда-исчисление. Его синтаксис и семантика*. Москва: Мир, 1985.
10. Дж. Бекус, "Алгебра функциональных программ: мышление функционального уровня, линейные уравнения и обобщенные определения", *Математическая логика в программировании*. Москва: Мир, 1991, с. 8–53.
11. В.Н. Редько, "Основы програмологии", *Кибернетика и системный анализ*, № 1, с. 35–57, 2000.
12. Р. Эшенхерст, *Лекции лауреатов премии Тьюринга*. Москва: Мир, 1993.
13. H.D. Curry, R. Hindley, and J.P. Seldin, *Combinatory Logic. Studies in Logic*. Amsterdam: North-Holland Co., 1972.
14. У. Дал, Э. Дейкстра, и К. Хоор, *Структурное программирование*. Москва: Мир, 1975.
15. Э. Дейкстра, *Дисциплина программирования*. Москва: Мир, 1978.
16. J. McCarthy, "Recursive Functions of Symbolic Expressions and Their Computation by Machine", *Communications of the ACM*, vol. 3, pp.184–195, 1960.
17. Н. Вирт, *Алгоритмы и структуры данных*. Москва: ДМК Пресс, 2010.
18. Р. Пенроуз, *Глава 2: Лямбда-исчисление Черча. Новый ум короля. О компьютерах, мышлении и законах физики.* Великобритания: Издательство Оксфордского университета, 2011.
19. В.В. Соколов, "Застосування функціональної та реляційної моделей в об'єктно-орієнтованому програмуванні", *Information Technology and Security*, № 1, с. 54–63, 2017.
20. О.В. Олецький та Є.В. Івохін, "Формалізація процедури формування динамічної рівноваги альтернатив у багатоагентному середовищі у процесах прийняття рішень більшістю голосів", *Кібернетика та системний аналіз*, № 1, с. 55–66, 2021.
21. Р. Мартин, *Часть II. Парадигмы программирования. Чистая архітектура*. СПб: Питер, 2018.
22. A. Appel, *Software Foundations Volume 3: Verified Functional Algorithms*. 2018. [Online]. Available: https://softwarefoundations.cis.upenn.edu/vfa-current/index.html
23. Miltner S. Padhi, T. Millstein, and D. Walker, "Data-driven inference of representation invariants", *41st ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2020)*. Association for Computing Machinery, New York, NY, USA, 2020, pp. 1–15. doi: https://doi.org/10.1145/3385412.3385967
24. V. Premtoon, J. Koppel, and A. Solar-Lezama, "Semantic code search via equational reasoning", *41st ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2020)*. Association for Computing Machinery, New York, NY, USA, 2020, pp. 1066–1082. doi: https://doi.org/10.1145/3385412.3386001
25. R. Ji, J. Liang, Y. Xiong, L. Zhang, and Z. Hu, "Question selection for interactive program synthesis", *41st ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2020)*. Association for Computing Machinery, New York, NY, USA, 2020, pp. 1143–1158. doi: https://doi.org/10.1145/3385412.3386025
26. Z. Guo et al., "Program synthesis by type-guided abstraction refinement", *41st ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2020)*. Association for Computing Machinery, New York, NY, USA, 2020, pp. 28. doi: https://doi.org/10.1145/3371080
27. M. Majthoub, M.H. Qutqut, and Y. Odeh, "Software Re-engineering: An Overview", *2018 8th International Conference on Computer Science and Information Technology (CSIT)*, Amman, 2018, pp. 266–270. doi: 10.1109/CSIT.2018.8486173.

Надійшла: 16.12.2020

INFORMATION ON THE ARTICLE

Igor V. Redko, ORCID: 0000-0002-3121-1412, National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”, Ukraine, e-mail: redkoigor@ukr.net

Petro O. Yahanov, ORCID: 0000-0001-7358-9846, National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”, Ukraine, e-mail: piteryahanov@gmail.com

Maksym O. Zylevich, ORCID: 0000-0003-1646-0557, National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”, Ukraine, e-mail: zila@meta.ua

РЕДУКЦИОННОЕ КОНЦЕПТИРОВАНИЕ ОРАКУЛЬНЫХ СХЕМ / И.В. Редько, П.А. Яганов, М.О. Зылевич

Аннотация. Работа направлена на развитие интерсубъективной парадигмы и активной роли субъекта в редукционном концептировании. Для этого общая оракульная схема концептирования конкретизирована взаимодополнением композиции и декомпозиции как экспликаций синтеза и анализа сущностей. Прагматико-обусловленное обогащение этого взаимодополнения осуществлено с привлечением композиционного программирования и именных моделей данных, функций и композиций. Рассмотрена оракульная схема редукции, смысл которой заключается в том, что она, опираясь на имеющиеся композиции, естественным образом имплементирует парадигму «разделяй и властвуй» в смысле активной роли субъекта в концептировании, поддерживая реальное взаимодополнение декомпозиционного и композиционного методов изучения концептирования. Приведены репрезентативные примеры редукционного концептирования, которые обосновывают технологию решения программистских задач.

Ключевые слова: программирование, интерсубъективная парадигма, оракульная схема, концептирование, редукция.

REDUCTION CONCEPTUALIZATION OF ORACLE SCHEMES / I.V. Redko, P.O. Yahanov, M.O. Zylevich

Abstract. This work is aimed at developing an intersubjective paradigm and an active role of the subject in reduction conceptualization. For this purpose, the general oracular scheme of conceptualization is concretized by the complementarity of composition and decomposition as explications of synthesis and analysis of entities. Pragmatically conditioned enrichment of this complementarity is carried out with the involvement of compositional programming and nominal models of data, functions, and compositions. The oracular scheme of reduction is considered, the meaning of which is that it, based on existing compositions, naturally implements the paradigm of “divide and conquer” in understanding the active role of the subject in conceptualization, supporting the real complementarity of decomposition and compositional methods of conceptualization. Representative examples of reduction conceptualization are shown, which substantiate the technology of solving programming problems.

Keywords: programming, intersubjective paradigm, oracle scheme, conceptualization, reduction.

REFERENCES

1. I. Redko and P. Yahanov, “Conceptual model of the technological environment of programming”, *KPI Science News*, vol.1, no.1, pp. 18–26, 2020. doi: 10.20535/kpisn.2020.1.197953.
2. I. Redko, P. Yahanov, and M. Zylevich, “Concept-Monadic Model of Technological Environment of Programming”, *2020 IEEE 2nd International Conference on System Analysis & Intelligent Computing (SAIC)*, Kyiv, Ukraine, pp. 125–130, 2020. doi: 10.1109/SAIC51296.2020.9239204.
3. I.V. Redko, D.I. Redko, and T.L. Zakharchenko, *Conceptual basis of programming.* Kyiv, Ukraine: Komprynt, 2016.
4. V.R. Karymov, “Arithmetic and hyperarithmetic computability with respect to constrained computations”, in *Mathematics and Mechanics*, no. 5, pp. 48–52, 2010.

5. M. Trakhtenbrot, “Mutation Patterns for Temporal Requirements of Reactive Systems”, *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Tokyo, 2017, pp. 116–121. doi: 10.1109/ICSTW.2017.27
6. A.I. Maltsev, *Algorithms and recursive functions*. Moscow, Russia: Nauka, 1965.
7. I. Basarab, N. Nykytchenko, and V. Redko, *Composite databases*. Kyiv, Ukraine: Lybid, 1992.
8. D.B. Buy, E.V. Shishatskaya, S. Fabunmi, and K. Mohammed, “Mathematical foundations of linearization algorithms: reflexive-transitive closure of binary relations”, in *Bulletin of V.N. Karazin Kharkiv National University*, no. 29, pp. 19–33, 2016.
9. H.P. Barendregt, *The Lambda Calculus. Its syntax and semantics*. Moscow, Russia: Mir, 1985.
10. J.W. Backus, “Algebra of functional programs: functional level thinking, linear equations and generalized definitions”, in *Mathematical logic in programming*. Moscow, Russia: Mir, 1991, pp. 8–53.
11. V.N. Redko, “Foundations of programmology”, *Kibernetika i sistemnyj analiz*, vol. 1, pp. 35–57, 2000.
12. R. Eshenkherst, *Turing Prize Winners Lectures*. Moscow, Russia: Mir, 1993.
13. H.D. Curry, R. Hindley, and J.P. Seldin, *Combinatory Logic. Studies in Logic*. Amsterdam: North-Holland Co., 1972.
14. U. Dal, E. Deikstra, and K. Khoor, *Structured programming*. Moscow, Russia: Mir, 1975.
15. E. Deikstra, *Discipline of programming*. Moscow, Russia: Mir, 1978.
16. J. McCarthy, “Recursive Functions of Symbolic Expressions and Their Computation by Machine”, *Communications of the ACM*, vol. 3, pp. 184–195, 1960.
17. N. Virt, *Algorithms and data structures*. Moscow, Russia: DMK Press, 2010.
18. R. Penrose, *Chapter 2: Church’s Lambda Calculus. The new mind of the king .About computers, thinking and the laws of physics*. United Kingdom: Oxford University Press, 2011.
19. V.V. Solokov, “Application of functional and relational models in object-oriented programming”, in *Information Technology and Security*, no. 1, pp. 54–63, 2017.
20. O.V. Olets’kyy and E.V. Ivokhin, “Formalization of the procedure for the formation of a dynamic balance of alternatives in a multi-agent environment in decision-making processes by a majority vote”, in *Cybernetics and Systems analysis*, no. 1, pp. 55–66, 2021.”
21. R. Martin, *Part II. Programming paradigms. Clean architecture*. Saint Petersburg, Russia: Piter, 2018.
22. A. Appel, *Software Foundations Volume 3: Verified Functional Algorithms*. 2018. [Online]. Available: https://softwarefoundations.cis.upenn.edu/vfa-current/index.html
23. Miltner S. Padhi, T. Millstein, and D. Walker, “Data-driven inference of representation invariants”, *41st ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2020)*. Association for Computing Machinery, New York, NY, USA, 2020, pp. 1–15. doi: https://doi.org/10.1145/3385412.3385967
24. V. Premtoon, J. Koppel, and A. Solar-Lezama, “Semantic code search via equational reasoning”, *41st ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2020)*.Association for Computing Machinery, New York, NY, USA, 2020, pp. 1066–1082. doi: https://doi.org/10.1145/3385412.3386001
25. R. Ji, J. Liang, Y. Xiong, L. Zhang, and Z. Hu, “Question selection for interactive program synthesis”, *41st ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2020)*. Association for Computing Machinery, New York, NY, USA, 2020, pp. 1143–1158. doi: https://doi.org/10.1145/3385412.3386025
26. Z. Guo et al., “Program synthesis by type-guided abstraction refinement”, *41st ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2020).* Association for Computing Machinery, New York, NY, USA, 2020, pp. 28. doi: https://doi.org/10.1145/3371080
27. M. Majthoub, M.H. Qutqut, and Y. Odeh, “Software Re-engineering: An Overview”, *2018 8th International Conference on Computer Science and Information Technology (CSIT),* Amman, 2018, pp. 266–270. doi: 10.1109/CSIT.2018.8486173.