УДК 681.3.06

І.В.РЕДЬКО, М.П.СЕМЕНЧЕНКО

Національний технічний университет України «КПІ»

Д.І.РЕДЬКО

Київський національний університет ім. Т.Г. Шевченка

СИСТЕМИ ПРОГРАМУВАННЯ: РЕДУКЦІЙНІ ЗАСАДИ

В рамках дефінітологічних досліджень систем програмування та спираючись на поняття композиційної та декомпозиційної систем (КС та ДКС відповідно) розглянуто процес породження композицій як основну складову процесу програмування. На основі поняття редукції в роботі побудовано ефективне предметне збагачення КС та ДКС до циклічних та дециклічних систем (ЦС та ДЦС відповідно). Взаємодоповнення останніх розглядається у свою чергу як редукційна дефінітна система (РДФС), що адекватно підтримує процес програмування як породження та застосування композицій. Розглянуто приклади застосування РДФС до вирішення репрезентативних задач

***Ключові слова**: дескриптивна система, дефінітна система, композиція, декомпозиція, редукція, породження композицій, застосування композицій, іменна функція, іменна множина.*

Згідно з [1-3] *програмування* розуміється *як породження та застосування композицій*. Тому глибоке проникнення в природу даних аспектів є необхідною умовою формування адекватних суті уявлень про природу власне програмування. Тим або іншим видам застосування композицій присвячено велику кількість ґрунтовних досліджень. Найбільш значимі результати отримані в рамках дослідження т.з. безтипового ламбда-числення та пов'язаних з ним аплікативних систем [4, 5]. Що ж стосується породження композицій, то його природа в значній мірі обумовлена взаємодоповненням відповідних композиційних та декомпозиційних систем [6]. Таким чином, розкриття глибинної суті породження композицій як ***головної мети*** даної статті в контексті взаємодоповнення КС та ДКС як ***об'єкта дослідження***, за необхідністю пов'язане з відповідним предметним збагаченням як самих КС та ДКС, так і їх взаємодоповнення. Основу даного предметного збагачення складає поняття редукції [7, 8]. Відповідні ж предметні збагачення КС та ДКС до циклічних і дециклічних систем та їх взаємодоповнення у вигляді редукційної дефінітної системи є ***предметом дослідження*** даної роботи.

Постановка завдання

Дана робота присвячена дослідженню програмування в аспекті породження композицій. Основна увага тут буде приділена дефінітологічній композиційних та декомпозиційних систем, а також їх важливих предметних збагачень – ЦС та ДЦС, як систем, що наповнюють реальним змістом розуміння програмування з точки зору породження та застосування композицій. Всі використані та невизначені в роботі терміни та поняття трактуються в сенсі [1].

Взаємодоповнення КС та ДКС як засіб породження композицій

Уведення КС та ДКС в їх взаємодоповнюючому зв'язку [6] суттєво збагачує розгляди програмування саме в аспекті породжень композицій. Ключовим об'єктом збагачення тут є двоєдиність цих породжень, яка проявляється у можливості прагматико-обумовленого вибору точки зору на породження композицій як на процес та як на наслідок останнього. У першому наближенні саме КС підтримує у взаємодоповненні біфуркаційність точок зору на породження композицій. При цьому, «процесну» точку зору максимально широко, аж до ототожнення в розглядах процесу з його наслідком, підтримує композиціал, який «поєднує» в собі знову ж таки у взаємодоповненні, як процес композитизації, так і його передумову. Композиція ж трактується тут як наслідок процесу. Вище вже зазначалось про неможливість повної об'єктивізації процесу (!) породження композицій та необхідність

залучення до розглядів суб'єкта. Рішення, що лежить «на поверхні» – повністю суб'єктивізувати композиціал, тобто, виключивши з розглядів сам процес породження, фактично постулювати його результат. Очевидно, що такі рішення можуть буди адекватно підтримані самою композиційною системою. Змістовно кажучи, це випадки, коли програмування у кращому випадку розглядається як застосування заданих композицій, а частіше навіть, ототожнюється з результатом такого застосування. Зрозуміло, що така трактовка програмування, точніше її абсолютизація, при всій її загальності йде у розріз зі згадуваним вище принципом обумовленості і вже тому не може розглядатись у якості «загального знаменника» розглядів програм та програмування. Причина у недостатній змістовності такого рішення. Тому необхідне його прагматико-обумовлене збагачення.

В першому наближенні таке збагачення повинно бути направленим на розкриття природи взаємодоповнюючого зв'язку КС та ДКС. Адже значимість взаємовпливу процесів декомпозиції та композиції не тільки в програмуванні, але й узагалі в розв'язанні будь-яких реальних задач не викликає сумніву (от лат. *divide et impera*). Природа зв'язку КС та ДКС, зокрема і з огляду на притаманну їй суб'єктивність, за необхідністю складно влаштована. Змістовно кажучи її зміст відображає прагматико-обумовлений, а значить суб'єкто-орієнтований зв'язок процесів породження композицій з процесами розв'язання задач. І якщо розв'язання будь-якої задачі як породження та застосування композиції (до підзадач) обумовлюється відповідною прагматико-обумовленою декомпозицією вихідної задачі, то сама декомпозиція, у свою чергу, обумовлюється відповідними (наявними) засобами композиціювання. Іншими словами, аналіз розв'язання різних задач показує, що пов'язані з ними композиції є структурованими сутностями, тобто їм притаманні ті чи інші структури. При цьому зрозуміло, що всі вони в своїй основі спираються на деякі базові композиційні структури, по відношенню до яких самі є похідними. Ці базові структури складають змістовно кажучи прагматико-обумовлений «загальний знаменник» бачення процесу розв'язання задачі суб'єктом її розв'язання.

З наведених змістовних роз'яснень випливає, що для будь-якої КС природа композиціалу як сутності, що підтримує породження композицій є обумовленою відповідною декомпозицією (результат процесу декомпозитизації). Побічно це відображено у визначеннях КС та ДКС, де композиціал та декомпозиція є сутностями одного роду – екстраад. З визначення ж ДКС стає очевидним, що декомпозиція як сутність, що обумовлює композиціал відповідної КС у свою чергу є обумовленою декомпозиціалом – сутністю, що підтримує декомпозитизацію задачі. З тих же змістовних роз'яснень, а також з того, що декомпозиціал та композиція є сутностями одного роду – інтроад слідує, що природа декомпозиціалу в свою чергу є обумовленою композиціями, що складають вищезгаданий «загальний знаменник» прагматико-обумовленого (суб'єкто-орієнтованого) розуміння процесу розв'язання задач, а значить є базою цього процесу. Такі міркування можна було б продовжувати як завгодно довго. Але з прагматичної точки зору можна цілком обмежитись вже наведеними. Таким чином, пара <КС, ДКС> адекватно підтримує прагматико-обумовлене програмування в частині суб'єкто-орієнтованого породження композицій. Композиційно-декомпозиційна взаємодія цих систем обумовлює логіку породження композицій. Що ж стосується предметної складової даного процесу, то її природа обумовлена багатьма чинниками, зокрема, наприклад, розглядуваною областю задач та відповідним до неї суб'єкто-орієнтованим вибором базових композицій.

Редукційне збагачення КС та ДКС

В [9, 10] однією з основних властивостей композицій програм названо їх адекватність уявленням розробника програм про способи програмотворення. Тому подальше збагачення композицій повинно проводитись з урахуванням даної вимоги. В першому наближенні можна стверджувати, що в теорії та практиці програмування загальновизнаним є розподіл композицій на два класи – ациклічні та циклічні композиції. Перший при цьому наслідує загально значимі для традиційної математики способи генезису програм як функцій, що ж стосується другого, то він, і це загальновизнано, відображає ту специфіку генезису функцій, яка є притаманною саме програмуванню. Тому подальші розгляди програмування з точки зору біополя <КС, ДКС> будуть тут акцентовані на циклічні структури генезису програм – композиції типу циклування. Конкретніше, у відповідності до [9, 11] такі композиції будуть розглядатись як алгебраїчні операції в класі іменних функцій.

У розглядах взаємодії КС і ДКС з урахуванням їх акцентації на циклічний характер композицій важливу роль відіграють редукції, як системостворююча основа декомпозиційних систем, що обумовлені композиціями циклування. У загальному випадку функцію g будемо називати редукцією функції f якщо $f = g; f$, де $;$ – представляє собою послідовне виконання функцій. Парадигмна значимість поняття редукції полягає в тому, що воно дозволяє адекватно викривати логіки задач, на основі яких різні реалізації їх у вигляді тих або інших процедур вирішення задач, різноманітних алгоритмів, моделей, програм і т.п. отримуються автоматично з гарантованою коректністю.

Загальність уведеного поняття редукції, що базується на поки що інтуїтивному розумінні як самих функцій, так і їх послідовного виконання індукує певну його відкритість у сенсі концепції ОС-System [8, 9, 12], а значить і недостатню змістовність. Для того, щоб подальші розгляди мали більш предметний характер необхідно провести відповідні збагачення змісту уведеного поняття редукції. Що стосується функцій, то тут будемо розуміти їх як іменні функції в сенсі [9, 11]. Послідовне виконання функцій, відповідно, конкретизуємо, наприклад, як операцію абстрактного множення функцій $\circ$ [9]. Крім того, під $*$ будемо розуміти бінарну алгебраїчну операцію ітерації, яка парі $< p, g >$, де p, g – функції, причому p – предикат, ставить у відповідність нову функцію, яку позначають $*(p, g)$ і значення якої на на аргументі d дорівнює першому з елементів послідовності $d, f(d), f(f(d)),\ldots$, для якого p приймає істинне значення, за умови, що для всіх попередніх елементів значення p визначено та є хибним [9]. Крім того, уведемо декілька корисних для подальших розглядів визначень, деякі з яких будуть за необхідністю суб'єктно-орієнтованими.

Образом множини P у функції $f : A \to B$, де $P \subseteq A$, будемо називати множину, яку будемо позначати $f(P)$, $f(P) \in B$ таку, що для будь-якого $a \in P$, для якого $f(a)$ є визначеним, $f(a) \in f(P)$.

Характеристикою функції $f : A \to B$ будемо називати таку множину $H_f \subseteq A$, для елементів якої значення $f(h)$, $h \in H_f$ є очевидним, або заздалегідь відомим. Зрозуміло, що так уведене поняття характеристики функції є суб'єкто-орієнтованим завдяки посиланням в ньому на очевидність або відомість. Але така суб'єктивність є цілком прийнятною в тому сенсі, що роль суб'єкта тут чітко локалізована. З формальної ж точки зору характеристика функції це підмножина її аргументів, дана ззовні.

Характеристику H_f функції $f: A \to B$ називатимемо *повною* або *F-характеристикою* f, якщо її образ у функції f – $f(H_f) = f(A)$. Уведені конкретизації дозволяють відобразити роль редукції у вигляді наступної теореми, справедливість якої випливає безпосередньо з вищенаведеного.

Теорема (необхідна умова редукційності). Якщо функція $f \equiv *(p, \hat{g})$, H_f – її характеристика, p – предикат такий, що $p(d) = \begin{cases} True, якщо\ d \notin H_f \\ False, якщо\ d \in H_f \end{cases}$, а функція g така, що для будь-якого аргументу d $g(d) = \begin{cases} \hat{g}(d), d \notin H_f \\ d, d \in H_f \end{cases}$ є редукцією функції f.

Аналогічні результати можуть бути отримані і для інших адекватних конкретизацій циклічних структур генезису функцій. Це означає, що поняття редукції носить загальнозначущий характер. В цьому сенсі воно являє собою концептуально-єдиний засіб розкриття логіки найрізноманітніших задач і в деякому сенсі носить універсальну природу. Однак у вищезазначеному сенсі використання поняття редукції носить все ж надто суб'єктивний характер. У вирішенні цієї проблеми важливу роль відіграють відповідні прагматико-обумовлені типізації поняття редукції. Ефективні редукції займають серед них чільне місце. Виходячи з того, що властивість ефективності не залежить від вибору сукупності базових композицій, дослідження її проведемо в контексті введених вище композицій абстрактного множення та циклування функцій. Зважаючи на вищезазначене це ніяк не вплине на загальність результатів.

Циклічна та дециклічна системи як предметне збагачення КС та ДКС

Проблема суб'єктивності застосування введеного поняття редукції полягає в тому, що хоча для будь-якої функції f, яка представляє собою застосування композиції циклування, тобто $f \equiv *(p, g)$ можна стверджувати, що функція g є її редукцією, однак стверджувати зворотне – з того, що g є редукцією f випливає, що $f \equiv *(p, g)$ для деякого предиката p в загальному випадку неможливо. Редукції, для яких останнє твердження є справедливим будемо називати *ефективними редукціями*. Дане визначення скоріше слугує прагматичною мотивацією дослідження властивості ефективності редукцій ніж змістовним збагаченням поняття редукції. Адже в даному випадку властивість позначена змістовно кажучи через її зовнішній прояв. Метою ж даного розділу статті є дослідження її внутрішньої структури. Характерною особливістю процесів, які обумовлені саме ефективними редукціями є те, що ефективна редукція функції f підтримує процес знаходження значення $f(a)$, де a – деякий аргумент функції f через покрокове зведення даної задачі до визначення значення $f(\hat{a})$, де $\hat{a}$ – аргумент f, для якого, по-перше, $f(a) = f(\hat{a})$ і, по-друге, значення $f(\hat{a})$ є заздалегідь відомим, тобто в конкретній прагматиці належить до характеристики функції f, тобто $\hat{a} \in H_f$. Сформульовані вимоги визначають змістовну суть ефективності редукцій. Таким чином, вирішення основної задачі даного розділу полягає в її уточненні.

Функцію $g: A \to A$ будемо називати (H, μ)-*ефективною* для деякого $H \subseteq A$ або просто *ефективною* у випадку коли явне зазначення H та μ не є необхідним, якщо існує натуральнозначна метрика $\mu: A \times A \to N$ така, що для будь-якого $a \in A$, для якого значення $g(a)$ визначено, існує таке

$h \in H$, що $\mu(g(a),h) \le \mu(a,h)$ та для будь-яких a,h існує таке $k \in N$ та $k \equiv k(a,h) : \mu(g^k(a),h) < \mu(a,h)$, де $g^k(a) \equiv \underbrace{g(g(...g(a)...))}_{k}$.

Характерною рисою так уведеної понятійної структури є те, що вона адекватним чином підтримує наведені вище змістовні вимоги до ефективності редукції. Стосовно даних понять безпосередньо випливає справедливість є наступної теореми.

Теорема (достатня умова редукційності). Для того, щоб функція f була результатом застосування композиції циклування, тобто, щоб $f \equiv *(p,g)$ достатньо, щоб функція g була ефективною редукцією функції f.

Цей результат відображає провідну роль ефективної редукції в породженні циклічних композиційних структур, основу яких складає композиція $*$. А саме показує, що декомпозитизація сутності, що індукована даною композицією циклування $*$, фактично зводиться до редуктизації цієї сутності – пошуку відповідної її ефективної редукції. У термінах ДФС сказане можна представити у вигляді наступної конкретизації взаємодоповнення КС та ДКС - циклічних та дециклічних систем.

Циклоціал ::= $*$-екстраада, що циклотизує сутність,
Циклозиція::= $*$-інтроада, що циклотизована циклоціалом,

Рециклоціал ::= $*$-інтроада, що редуктизує сутність,
Рециклозиція::= $*$-екстраада, що редуктизована рециклоціалом,

де циклотизує (циклотизована) розуміється як $*$-композитно інтроспектує ($*$-композитно інтраспектована), а редуктизує (редуктизована) – як де-$*$-композитно екстраспектує (де-$*$-композитно екстраспектована).

Приклад програмування знаходження найбільшого спільного дільника (НСД) двох натуральних чисел

Перш за все необхідно відзначити, що наведена в назві розділу постановка задачі є досить неповною. Тому необхідно сформулювати додаткові вимоги, що повинні визначити основні аспекти результуючої програми. Головним серед них є прагматичний аспект, що задає характер використання результату програмування. Саме він визначає відношення між програмою як результатом програмування та тими, хто її використовує, зокрема характеризує виконавця, на якого розрахована програма, засоби обробки даних, які йому доступні. Другий аспект – семантичний, пов'язаний з визначенням змісту програм, тобто того, що вони повинні здійснювати. В першу чергу необхідно вказати вихідні дані програми, її результати та відповідність між ними. По цим двом аспектам домовимось, що задля прозорості викладення та з огляду на простоту задачі, виконавець задовольняє вимогам, що були наведені в попередньому розділі. Тобто, програма буде з семантичної точки зору розумітись як абстрактна іменна функція, а щодо засобів генезису, то вони обмежуються введеними вище абстрактними композиціями $*$, $\circ$ та абстрактною композицією галуження IF [9, 11]. Крім того, на даному етапі виконавця влаштовує деталізація задачі на рівні доступні в якості елементарних операції знаходження залишку від ділення двох натуральних чисел $\mathrm{mod}(m,n)$, логічні операції – $\le,\ge,=,\ne,<,>$. Що ж стосується третього, синтаксичного, аспекту, то тут йому буде приділено мінімальну увагу, в

першу чергу, задля скорочення об'єму статті, а також з огляду на те, що ґрунтовній проробці його присвячено величезну кількість публікацій.

Стосовно прикладу, у якості вихідних даних та результату можна взяти іменну множину $\{(M,m),(N,n)\}$, де M,N – імена, а m,n – відповідні денотати. Відповідність даних і результатів задається іменною функцією $НОД_{M,N}:\{(M,m),(N,n)\}\to\{(M,nod(m,n)),(N,0)\}|_{m,n\in N}$ [9, 11], де $nod(m,n)$ – позначає число, яке є найбільшим спільним дільником (НСД) натуральних чисел m,n. Виходячи з того, що функція не є безпосередньо доступною виконавцю, для вирішення основної задачі необхідно знайти ефективну редукцію функції $НОД_{M,N}$. Очевидно, що характеристикою функції $НОД_{M,N}$ виступати множина $H\equiv\{\{(M,m),(N,0)\}\,|\,m\in N\}$. Що ж до натурально значної метрики μ, то її природа обумовлена наведеною нижче характеристичною властивістю функції $НОД_{M,N}$:

$$НОД_{M,N}(\{(M,m),(N,n)\})=\begin{cases}\{(M,m),(N,n)\}, якщо\ n=0,\\ НОД_{M,N}(\{(M,n),(N,\mathrm{mod}(m,n))\}),\end{cases}$$

а сама метрика може задаватись, наприклад, таким чином:

$\mu(\{(M,m_1),(N,n_1)\},\{(M,m),(N,n_2)\})\equiv|\,n_1-n_2\,|$.

Враховуючи викладене є очевидним, що функція $g:\{(M,m),(N,n)\}\to\begin{cases}\{(M,m),(N,n)\}, якщо\ n=0\\ \{(M,n),(N,\mathrm{mod}(m,n))\}, якщо\ n>0\end{cases}\Bigg|_{m,n\in N}$ є (H,μ)-ефективною редукцією функції $НОД_{M,N}$. Таким чином, $НОД_{M,N}\equiv *(IF([N\Rightarrow]\neq 0),\hat{g})$, де $[N\Rightarrow]$ – операція розіменування: $[N\Rightarrow](d)=n, якщо\ (N,n)\in d$ [9,11], а $\hat{g}$ така, що $\hat{g}(\{(M,m),(N,n)\})=\{(M,n),(N,\mathrm{mod}(m,n))\}|_{m,n\in N\&n>0}$ і вихідна задача вирішена в один крок. Тепер дещо ускладнимо задачу та припустимо, що обраний рівень деталізації виконавця вже не влаштовує тому, що насправді функція mod не є елементарною для нього. Натомість доступними є арифметичні операції додавання та різниці двох натуральних чисел. Виникає необхідність роз'яснити останньому суть функції mod у термінах, які зрозумілі виконавцю. Потрібно здійснити ще один крок редуктизації, але вже функції mod.

У якості вихідних даних та результату можна взяти іменну множину $\{(M,m),(N,n)\}$, де M,N – імена, а m,n – відповідні денотати. Відповідність даних і результатів задається відповідно іменною функцією $MOD_{M,N}:\{(M,m),(N,n)\}\to\{(M,\mathrm{mod}(m,n)),(N,n)\}|_{m,n\in N}$. Виходячи з зазначеного, для вирішення основної задачі необхідно знайти ефективну редукцію функції $MOD_{M,N}$. Очевидно, що характеристикою функції $MOD_{M,N}$ виступати множина $H\equiv\{\{(M,m),(N,n)\}\,|\,m\in N\}|_{m,n\in N\&n>m}$, а виходячи з характеристичної властивості функції mod: $\mathrm{mod}(m,n)=\mathrm{mod}(m-n,n)|_{m,n\in N\&m>n}$, натурально значна метрика μ очевидно може бути представлена так: $\mu(\{(M,m_1),(N,n_1)\},\{(M,m_2),(N,n_2)\})=|\,m_1-m_2\,|$. Тоді функція

$g_{\text{mod}} : \{(M,m),(N,n)\} \to \begin{cases} \{(M,m),(N,n)\}, \text{яякщ } n > m \\ \{(M,m-n),(N,n)\}, \text{яякщ } n < m \end{cases}\Big|_{m,n\in N}$ очевидно є (H,μ)-ефективною редукцією функції $MOD_{M,N}$. І таким чином, $MOD_{M,N} \equiv *(IF([M \Rightarrow] > [N \Rightarrow]), \hat{g}_{\text{mod}})$, де $[M \Rightarrow]$, $[N \Rightarrow]$ та $\hat{g}_{\text{mod}}$ розуміються аналогічно до вищезазначеного. Це дає можливість адекватної умовам задачі подальшої деталізації функції $НОД_{M,N} \equiv *(IF([N \Rightarrow] \neq 0), \hat{g})$, у першу чергу, за рахунок конкретизації функції $\hat{g}$. А саме, $\hat{g} : \{(M,m),(N,n)\} \to \{(M,n),(N,\text{mod}(m,n))\}|_{m,n\in N \& n>0}$ відповідно до зазначеного може бути представлена, наприклад, так: $\hat{g} \equiv [\Rightarrow R]([\overbrace{M \Rightarrow](*(IF([M \Rightarrow] > [N \Rightarrow]), \hat{g}_{\text{mod}}))}) \circ [\Rightarrow M]([N \Rightarrow]) \circ [\Rightarrow N]([R \Rightarrow])$ або, в більш звичній Паскаль-подібній формі: $\hat{g} \equiv MOD_{M,N};\ R := M;\ M := N;\ N := R.$

Висновки

Рішення будь-якої задачі суть композиція рішень її підзадач. Визначальна значимість композиційної проблематики в інформатико-технологічній області і особливо, у програмуванні сьогодні загальновизнана. При цьому, сучасний її стан характерний тим, що застосовувані тут традиційні засоби вирішення задач носять явно виражену экстенсивну природу. Стосовно методів та засобів програмування це проявляється в тому, що усі вони підтримують стратегію композиціонування лише в частині нотації результатів програмування як породження композицій та застосування їх до рішень окремих підзадач. І ніяким чином не орієнтовані власне на саме програмування. Яскравим представником цього підходу є, наприклад, т.зв. модульне програмування, у якому основна увага приділена стандартизації власне модулів, зокрема, властивих їм засобів комутації як універсального засобу композиціонування і, разом з тим, практично нівельована підтримка процесів породження та застосування самих композицій. Однак для реальних задач не стільки важлива потенційна можливість їх вирішення, що без сумніву забезпечується традиційними, зокрема модульними, засобами програмування, скільки ефективна, тобто *нерозривно пов'язана* з прагматикою розв'язуваної задачі, підтримка процесів пошуку таких рішень. Останні ж, як показано в даній роботі, нерозривно пов’язані з породженням та застосуванням композицій. В якості таких інструментів програмування в статті як раз і розглядаються редукції – адекватні прагматиці задач засоби породження та застосування композицій. Тому створення інструментальних комплексів, що підтримують побудову прагматико-обумовлених редукційних середовищ програмування є ключовим завданням в композиційній проблематиці СП. Запропонований у роботі підхід до дослідження СП за допомогою залучення в розгляд взаємодії полюсів біполя <КС, ДКС> дозволяє реально, а не номінально підтримати взаємодоповнення процесів вирішення інформатико-технологічних проблем з їхніми результатами й, отже, не просто забезпечити потенційну можливість рішення програмістської задачі, але й, що найбільш важливо, підтримати процес покрокового породження відповідного композиційного середовища.

Список використаної літератури

1. Редько В.Н. Дескриптологические основания программирования // Кибернетика и системный анализ. – 2002. – № 1. – С.3 – 19.

2. Редько В.Н. Основания дескриптологии // Кибернетика и системный анализ. – 2003. – № 5. – С.16 – 36.

3. Редько В.Н.,Редько И.В, Гришко Н.В. Дескриптивные системы: концептуальный базис// Проблеми програмування. – 2006. – № 2–3. – С. 75–80.

4. Редько В.Н.,Редько И.В, Гришко Н.В. Дескриптивные системы: концептуальный базис// Пробл. програмув. – 2006. – № 2–3. – С. 75–80.

5. Барендрегт Х. Ламбда-исчисление. – М.: Мир. – 1985. – 606 с.

6. Редько В.Н.,Редько И.В, Гришко Н.В. Дефинитологические основания сущностной платформы// Пробл. програмув. – 2012. – № 2 – 3. – С. 5 – 19.

7. Редько В.Н., Гришко Н.В., Редько И.В. Экспликативное программирование в среде логико-математических спецификаций// Труды Первой международной научно-практической конференции по программированию УкрПРОГ'98 (доклад).– Киев. –1998. – с.71–76

8. Редько І.В. Процесологічні середовища системного аналізу // Систем. дослідж. та інформ. технології. – 2004. – № 4. – С. 124 – 140

9. В.Н. Редько, І.А. Басараб, М.С. Нікітченко Композиційні бази даних. – К.:Либідь,1992.–192 с.

10. Редько И.В. Теория дескриптивных сред и ее применения. Докт.диссерт. – К.: НТУУ «КПІ», 2008. – 403 с.

11. Редько В.Н. Композиции программ и композиционное программирование // Программирование. – 1978. – № 5. – С. 3 – 24.

12. Редько В.Н., Редько И.В, Гришко Н.В. Программологические основания сущностной платформы// Проблеми програмування. – 2008. – № 2–3. – С. 72–78.

Стаття надійшла до редакції 11.07.2012

Системы программирования: редукционные основания

Редько И.В., Семенченко М.П.
Национальный технический университет Украины «КПИ»
Редько Д.И.
Киевский национальный университет им. Т.Г. Шевченко

В рамках дефинитологических исследований систем программирования и, опираясь на понятия композиционной и декомпозиционной систем (КС и ДКС соответственно), рассмотрен процесс порождения композиций як основной составляющей процесса программирования. На основании понятия редукции в роботе построено эффективное предметное обогащение КС и ДКС до циклических и дециклических систем (ЦС и ДЦС соответственно). Взаимодополнение последних рассматривается, в свою очередь, как редукционная дефинитная система (РДФС), которая адекватно поддерживает процесс программирования как порождения и применения композиций. Рассмотрены примеры применения РДФС к решению репрезентативных задач.

Ключевые слова: дескриптивная система, дефинитная система, композиция, декомпозиция, редукция, порождение композиций, применение композиций, именная функция, именное множество.

The systems of programming: reductive foundations

Redko I., Semenchenko M.
National Technical University of Ukraine «Kyiv Polytechnic Institute»
Redko D.
Taras Shevcchenko National University of Kyiv

Within the scope of definitological exploration of system of programming and based on notions of compositional and decompositional systems (CS and DCS accordingly) the process of creation of compositions as the main part of programming process has been considered. On the base of notion of reduction an effective

object enrichment of CS and DCS to loop and de-loop systems (LS and DLS accordingly) has been built within this article. The mutual supplementation of the last-mentioned are considered as reduction definitical system (RDFS), that adequately supports programming process as creation and application of compositions. As an example RDFS has been applied to solve representative problems.

Keywords: descriptive system, definitive system, composition, decomposition, reduction, process of creation of compositions, application of compositions, nominal function, nominal set.