УДК 681.3.06; 519.7

С. О. Ковальов
Харківський національний університет радіоелектроніки,
кафедра програмного забезпечення ЕОМ

ФОРМАЛЬНІ МОДЕЛІ В ОБ'ЄКТНО-ОРІЄНТОВАНІЙ ПАРАДИГМІ

Attempt to formalize semantics of object-oriented paradigm is given. There are used predicate transformers, multidimensional spaces, spaces of states.

Наведено спробу формалізації семантики об'єктно-орієнтовної парадигми засобами перетворювачів предикатів, багатовимірного простору, простору станів обчислювань.

Вступ

Вивчення мов програмування схоже з вивченням природних мов. У програмуванні вирізняють дві точки зору на мови програмування та самі програми: синтаксис та семантику. Синтаксис мови програмування – це форма. Семантика – це сенс його виражень, операторів та програмних одиниць.

Наприклад, оператор if мови програмування C має такий синтаксис:

if (<вираз>) <оператор>;

Семантика полягає у тому, що якщо <вираз> має значення „Істина", то буде виконано наведений оператор.

Для опису синтаксису розроблено багато потужних формалізмів. Наприклад, форма Бекуса-Наура (Backus-Naur Form) є практично стандартом. Для семантики ще не знайдено загальноприйнятих методів запису.

Описувати сенс програм потрібно з кількох причин. Програмістам потрібно знати семантику операторів, які вони використовують. Розробникам компіляторів потрібно визначити семантику мов, для яких вони створюють компілятори. На опис формальної семантики також спирається доказ коректності програм.

Потрібно описати формальну семантику для наведеної на рис. 1 структури [0].

Сформулюємо критерії до математичного апарату:

1. Описувати семантику змістовних положень.
2. Бути конструктивним, щодо отримання нових знань та дослідження властивостей моделей та об'єктів, що описуються.
3. Мати максимально доступну та зрозумілу форму.
4. Прямувати до цілі автоматичного (комп'ютерного) отримання результатів, їх інтерпретації та подання.

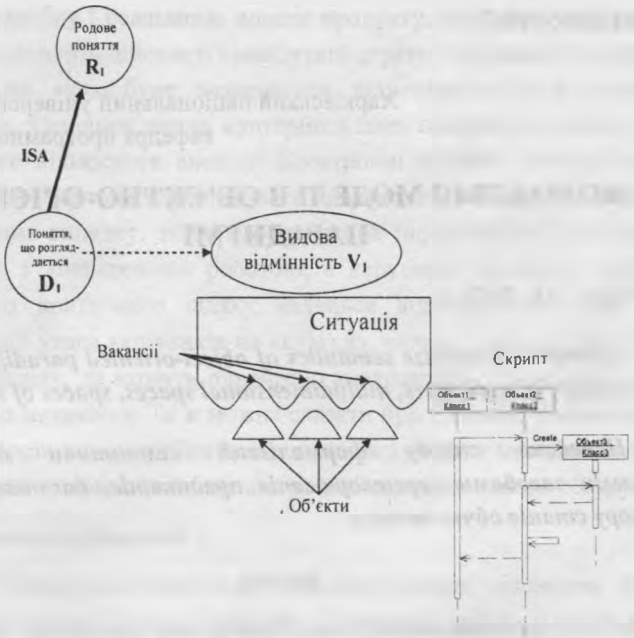

Рис 1. *Розкриття родовидового визначення в „осколок" та „ситуацію"*

Алгебра ситуацій

Для формалізації розкриття родовидового визначення за допомогою зведення до варіантів використання даного поняття у вигляді ситуацій скористаємося математичним апаратом теорії множин.

Сама ситуація має таку внутрішню структуру:

$$\zeta \overset{def}{=} \langle R', S'(R') \rangle,$$

де R' – множина валентностей (ролей), що задіяні у ситуації ζ, така, що $R' \subseteq R$; S' – скрипт (сценарій), що специфікує взаємодію об'єктів, які заповнюють валентності R', такі, що $S' \subseteq S$.

Родовидове визначення поняття δ, що розглядається, розкриваємо у структуру, що характеризується:

- Родовидовим відношенням поміж поняттям δ, що визначається, та його родом ρ

$$(\delta, \rho) \in \Theta$$

- Відображенням поняття δ, що визначається, до множини ситуацій R

$$Sit : \Psi \longrightarrow Z$$

Математичний апарат теорії множин є досить примітивним засобом опису результатів. Але він може бути базисом до введення більш конструктивних формалізмів.

Формалізація опису ієрархії компонент моделі

Класифікація багатовимірних спостережень.

Багатовимірні випадкові величини

Згідно з визначенням [6, с.12] *багатовимірна випадкова величина* є «набором кількісних ознак певного фізичного змісту

$$X = \begin{pmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(p)} \end{pmatrix}$$

значення кожного з них схильні до безконтрольного розкиду при повторі даного процесу, спостереження, експерименту».

Статистичні методи оперують *багатовимірними спостереженнями* X_i багатовимірної випадкової величини, що досліджується, як результатами виміру ознак $x^{(j)}$ на і-м предметі, що розглядається.

Вичерпні відомості про поведінку випадкової величини X надає ймовірнісна міра P(ΔS), наприклад, у вигляді функції щільності розподілу ймовірностей f(U).

$$f(U) = \lim_{\substack{n\to\infty \\ V_\varepsilon \to 0}} \frac{\upsilon_n(\varepsilon(U))}{nV_{\varepsilon(U)}},$$

$$P(\Delta S) = \int_{\Delta S} f(U)dU$$

де інтегрування виконуються за змінними $u^{(1)}$, …, $u^{(p)}$ по усій заданій ділянці ΔS.

Геометрично багатовимірні спостереження можна інтерпретувати як точки у *p*-вимірному просторі. Геометрична близькість точок у такому просторі позначає близькість фізичних станів об'єктів. Важливим і вирішальним фактором є вибір метрики (визначення та оцінювання відстаней між точками).

Залежно від типу вихідної інформації для побудови математичної моделі розглядають доцільність та ефективність використовування різних методів. Наприклад, кластер-аналіз, непараметричні евристичні методи класифікації, параметричні методи класифікації, методи розщеплення суміші та інші [4].

Адаптація до потреб об'єктно-орієнтованої парадигми (ООПар).

Математично-статистичні методи можуть бути застосовані до класифікації об'єктів, заданих набором кількісних ознак. Геометрична інтерпретація цього – розподіл точок у багатовимірному просторі по ділянках.

У об'єктно-орієнтованій парадигмі можна знайти такі відповідності:

- багатовимірна випадкова величина – клас;
- генеральна сукупність – домен класу (сукупність усіх екземплярів);

- спостереження – стан екземпляра.

В ООПар програма репрезентується як множина об'єктів (екземплярів чітко визначених класів), які обмінюються повідомленнями. Основна задача теорії класифікації багатовимірних спостережень у наведеній інтерпретації є з'ясування поточного (динамічного!) стану екземпляра. Моделювання життєвого циклу об'єктів, їх динамічній поведінки важливо для ООПар. Але акцент ставиться на специфікацію цієї поведінки, чітке завдання станів та правил їх зміни. Задача практично протилежна.

Актуальне знаходження всіх характеристик об'єктів, точніше, їх розподіл по них (в [0] названо «ідентифікацією класів»). В термінах теорії багатовимірних спостережень – зіставлення *різних* багатовимірних величин або, для геометричної інтерпретації, різних просторів, навіть з різними вимірами.

Але сама ідея подання класу як багатовимірного простору *плодотворна*. Виміри це атрибути, а значення атрибутів – характеристика об'єкта за даним виміром. В ООПар атрибути самі є класи, а їх значення – екземпляри класів атрибутів. Так спускаються до «елементарних» типів (класів в ОО мовах програмування), які легко відображаються в апаратних засобах обчислювальної платформи.

Корисним буде узагальнення елементарних типів до єдиного представлення класу. Будемо розуміти клас як багатовимірний простір, де виміри є інші класи нижчого рівня абстракції. Тоді, наприклад, тип «ціле» (int у мові програмування C++, Integer у Object Pascal) це одновимірний простір, у якому усі екземпляри (значення) строго упорядковані. Будемо називати такі «елементарні простори» класами першого рівня. З класів одного (i-го) рівня за допомогою *композиції* формуються простори вищого (i+1 -го).

$$K^j = < A_l^{j-1} >,\ l = \overline{1,r}$$

де A_l^{j-1} – l-й атрибут r-вимірного класу j-го рівня.

Індекс атрибута вказує на факт композиції. Зменшення індексу позначає зниження рівня розгляду партитивної (ціло-часткової) ієрархії.

Атрибут сам є класом, звідки має сенс узагальнити попередній вираз:

$$K^{j,v} = < K_l^{j-1,w} >,\ l = \overline{1,r}$$

де $K_l^{j-1,w}$ – l-й атрибут r-вимірного класу j-го рівня, який є класом (j-1)-го рівня в ієрархії композиції; v, w – унікальні індекси класів.

Унікальна ідентифікація класів необхідна для:

- урахування перемінної кількості класів на рівнях;
- принципово необмеженої можливості використовувати один і той же клас на різних рівнях партитивної ієрархії.

Описаний апарат формалізує структурну сторону класу. Крім того, потрібно описати поведінку класу, його динамічні характеристики, ті можливості, які він надає до взаємодії.

Формалізація процесуальних моделей

Засоби опису дій

Від процедурної до об'єктно-орієнтованої парадигми програмування

Крім опису структури абстрактних типів даних (члени-дані у класах), невід'ємною частиною *об'єкта* є опис його поведінки. Цей опис у мовах програмування виражається у декларації застосовних до цього об'єкта функцій, тобто *методів* даного класу (кажуть ще про специфікацію *повідомлення*, на які він може адекватно реагувати) та їх реалізацію. Реалізація методу полягає як в організації ліпшої взаємодії низького рівня між об'єктами, так й у виразі алгоритму виконання дії. Та частина мов програмування, яка й використовується безпосередньо для опису алгоритмів і взаємодій об'єктів, називається *засобами опису дій*.

Питання семантики та синтаксису засобів опису дій для структурних імперативних мов детально розглядаються в [9, 10, 11], засоби формального опису семантики – в [4, 5, 6, 12, 13]. Дослідження та розробка засобів формального опису дій в об'єктно-орієнтованій парадигмі базуються на досягненнях для структурних імперативних мов [14]. Далі розробляються формалізми, які інтегрально враховують особливості об'єктного підходу до побудови програм.

Формалізми доказу правильності програм

Описати семантику оператора мови програмування [10, 13] можна за допомогою *специфікацій програми*, які базуються на стані обчислювань до і після виконання оператора. *Стан обчислень* [13, с.110] в будь-який момент часу розуміється як «значення усіх змінних у цей момент», яке залежить від початкових значень, заданих при запуску програми і усіх попередніх дій.

Семантика оператора S (підпрограми або програми як послідовності операторів) описується так:

$$\{P\}\ S\ \{Q\},$$

де P – предикат, який описує стан обчислювань *до* виконання S; Q – предикат, який описує стан обчислювань *після* виконання S.

При цьому P повинен бути істинним до виконання і називається передумовою S. Q повинен бути істинним після виконання S і називається післяумовою S.

Використаємо апарат логіки предикатів для опису операторів дії за допомогою *правил виводу* [1, 3].

Для оператора присвоювання:

$$\{P_X^E\}\ X := E; \{P\},$$

де P – логічний вираз; P_X^E – вираз, який ми отримаємо унаслідок підстановки виразу E замість усіх вільних надходжень змінної X в P; X – змінна або інший припустимий об'єкт даних; E – вираз відповідного змінній типу.

Для порожнього оператора:

$$\{P\}\ ;\ \{P\}.$$

Для оператора зупинки, який перериває роботу програми:

{P} stop; {false}

Для послідовності операторів $S_1\ S_2\ \dots\ S_n$:

$$\frac{\forall i\ (1 \le i \le n \Rightarrow \{P_{i-1}\}\ S_i\ \{P_i\})}{\{P_0\}\ S_1\ S_2 \dots S_{n-1}\ \{P_{n-1}\}},$$

де $S_1, S_2, \dots S_n$ – оператори або послідовності операторів.

Для умовного оператора:

$$\frac{\{P \& B\}\ S_1\ \{Q\}, \{P\ \&\ not\ B\}\ S_2\ \{Q\}}{\{P\}\ if\ B\ then\ S_1\ else\ S_2\ end\ if; \{Q\}},$$

де B – логічний вираз (тобто має тип Boolean).

Для скороченої форми умовного оператора:

$$\frac{\{P \& B\}\ S\ \{Q\},\quad (P \&\ not\ B)\ \Rightarrow\ Q}{\{P\}\ if\ B\ then\ S\ end\ if; \{Q\}}$$

Для загальної форми умовного оператора:

$$\frac{\forall k(1 \le k \le n \Rightarrow (\{P \& \forall i(1 \le i \le k \Rightarrow not\ B_i) \& B_k\}\ S_k\ \{Q\})), \{P \& \forall i(1 \le i \le n \Rightarrow not\ B_i)\}\ S_n\ \{Q\}}{\{P\}\ if\ B_1\ then\ S_1\ elsif\ \dots\ else\ S_n\ end\ if; \{Q\}}$$

Для оператора вибору:

$$\frac{\forall i(1 \le i \le n \Rightarrow (\{P \& (E = v_i)\}\ S_i\ \{Q\})), \{P \& \forall i(1 \le i \le n \Rightarrow (E \ne v_i))\}\ S_n\ \{Q\}}{\{P\}\ case\ E\ of\ v_1 : S_1 \dots others : S_n\ end\ case; \{Q\}},$$

де E – якийсь вираз; v_i – константи того ж типу, що й значення виразу.

Для оператора вибору Дейкстри:

$$\frac{\forall i\ (1 \le i \le n\ \Rightarrow\ \{P \&\ B_i\}\ S_i\ \{Q\})}{\{P \&\ BB\}\ if\ B_1 : S_1\ B_2 : S_2 \dots B_n : S_n\ end\ if; \{Q\}},$$

де $BB = \exists i\ (1 \le i \le n \&\ B_i)$.

Для оператора циклу while-do:

$$\frac{\{P \& B\}\ S\ \{P\}}{\{P\}\ while\ B\ do\ S\ end\ do; \{P \&\ not\ B\}}.$$

Для оператора циклу do-until:

$$\frac{\{P\}\ S\ \{Q\}, (Q \&\ not\ B) \Rightarrow P}{\{P\}\ do\ S\ until\ B; \{Q \& B\}}.$$

Для оператора вибору Дейкстри:

$$\frac{\forall i\,(1 \le i \le n \Rightarrow \{P \,\&\, B_i\}\; S_i\; \{P\})}{\{P\}\; do\; B_1 : S_1\; B_2 : S_2\; \ldots\; B_n : S_n\; end\; do;\; \{P \,\&\, not\; BB\}},$$

де $BB = \exists i\,(1 \le i \le n \,\&\, B_i)$.

Для оператора for:

$$\frac{\{(mun'FIRST \le X \le mun'LAST) \,\&\, P([mun'FIRST\; ..\; X))\}\; S(X)\; \{P([mun'FIRST\; ..\; X])\}}{\{P([mun'FIRST\; ..mun'FIRST])\}\; for\; X : mun\; do\; S(X)\; end\; do;\; \{P([mun'FIRST\; ..mun'LAST])\}},$$

де S(X) – послідовність операторів, при виконанні яких використовується значення змінної X, причому воно не змінюється; P(w) – предикат відносно наступних інтервалів w – [a..b] = {X | a ≤ X ≤ b}; [a..b) = {X | a ≤ X < b}; (a..b] = {X | a < X ≤ b}. При цьому припускається, що кожне виконання S розширює інтервал, на якому істинне P: $\{P([mun'FIRST\; ..\; X))\}\; S(X)\; \{P([mun'FIRST\; ..\; X])\}$ и наступній предикат є істиною $P([mun'FIRST\; ..\; mun'FIRST))$.

Поширення формалізмів на об'єктно-орієнтований підхід

Порівняно з процедурним підходом на низькому рівні додаються наступні особливості.

Перша особливість

Акцент переноситься на використання процедур та функцій, які належать одному модулю – методів класу ОО парадигми. Витрати на переключення стека, які виникають при цьому, або вважаються виправданими (за рахунок збільшення структурованості і зрозумілості), або оптимізуються різними методами.

Переважно використовуються два методи оптимізації зайвих викликів. При застосуванні *першого* здійснюється підміна на машинному рівні виклику методу його повним текстом. Відбувається поліпшення часових характеристик за рахунок збільшення розміру програми. Перший метод доцільно використовувати для оптимізації викликів коротких и часто тривіальних методів. Тенденція розвитку апаратних засобів проявляється у постійному зменшенні вартості усіх носіїв інформації. Таким чином, суб'єктивна міра «стислості» та «тривіальності» методів класів буде збільшуватись, що відкриває нові можливості застосування розглянутого методу оптимізації зайвих порівняно з процедурним підходом викликів.

Другий метод полягає у використанні узагальненого програмування за допомогою *шаблонів* (template). На рівні тексту програми створюється шаблон функції або класу цілком. Під час складання програми шаблон заміняється кодом низького рівня, який генерується автоматично. Головним недоліком є складність зрозуміння (і як наслідок, використання) цього механізму. Також потрібно відзначити наявність обмежень на використання засобів мови під час створення шаблонів.

Друга особливість

Головна мета програмної системи досягається не виконанням алгоритму як у процедурній парадигмі, а взаємодією екземплярів класів. Ця взаємодія відбувається за допомогою надсилання повідомлення (запитів на виконання дій) одного екземпляра

іншому. В наслідок цього утворюється складний, не лінійний малюнок виконання методів класів, які знаходяться на різних рівнях абстракції.

У такому разі про осмислений алгоритм можна говорити у межах одного методу, який складений з викликів методів, що мають нижчий рівень. Тобто є подібна композиція методів на рівнях класів, що мають високий рівень, кооперацій підсистем і програмної системи у цілому

Третя особливість

Сам метод відрізняється від традиційного (процедурного) розуміння функції або процедури. Методу, крім глобальних даних, доступні локальні для обраного екземпляра дані. При цьому для кожного екземпляра ці дані різні.

Розглянуті особливості підкреслюють необхідність формалізації у першу чергу характеристик поведінки класів. Також задають напрям такої формалізації – методи класів.

Узагальнюючи попередні формалізми, запропонуємо:

- описувати атрибути (структуру класу) у вигляді сукупності двох методів – акцессрів – метод «читання» і метод «запису»;
- описувати метод як процедуру, яка сприймає додатковий параметр, який дозволяє отримати данні конкретного екземпляра;
- будь-який метод потенційно змінює стан екземпляра, через це формальне узагальнення методу у вигляді процедури повинно повертати новий стан екземпляру.

Таким чином, клас моделюється багатовимірним простором, стан екземпляра – точка багатовимірного простору, метод – функтор, що переводить екземпляр з одного стану в інший, тобто який відображає одну точку багатовимірного простору в іншій.

Внутрішнє наповнення методу полягає або у композиції методів, що мають нижчий рівень, або (на найнижчому рівні) у перетворювачі предикатів у стилі Е. Дейкстри. При цьому якраз стан обчислень й описується як стан екземпляра. При розгляді програмної системи у цілому її можна узагальнить як клас таких програм, де програма, що виконується – це конкретний екземпляр, стан обчислень – стан екземпляра.

Приклад

Маємо точку у просторі. Для спрощення будемо розглядати двовимірний простір тільки з цілими координатами.

Формалізуємо клас Point (точка). Точка складається з вимірів (Dimension). Сам вимір є надбудовою над типом цілих чисел. Тобто атрибути вимір_X, вимір_Y мають тип Integer.

```
class Integer {
private:
  int Value;
public:
  Integer(int aValue){Value = aValue;};
  void SetValue(int i){Value = i};
  int GetValue(){return Value;};
};
class Point {
```

```
private:
  Integer Dim_X, Dim_Y;
public:
  void SetX(Integer aX){Dim_X.SetValue(aX);};
  int GetX(){return Dim_X.GetValue();};
  void SetY(Integer aY) {Dim_Y.SetValue(aY);};
  int GetY(){return Dim_Y.GetValue();};
};
```

Розглядаючи класи і стани екземплярів як багатовимірні простори, маємо наочне графічне представлення.

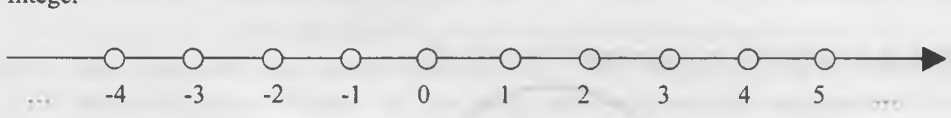

Рис 2. *Одновимірний простір класу Integer.*

Математичне поняття цілого числа має нескінченну множину значень, не обмежену ні зліва, ні справа. У дискретних цифрових ЕОМ (які переважають нині) існують обмеження, зв'язані з пам'яттю. Число необхідно якось зберігати. Через це цілі типи даних обмежуються конкретним діапазоном, а дійсні (реальні) числа надаються з якимсь наближенням. Ступінь наближення оцінюють кількістю значущих (точних) цифр, для дійсних також відстанню між двома точно наданими числами (через особливості надання у формі множника і мантиси ця відстань тим більше, чим більше по абсолютному значенню число).

Таким чином, за допомогою конкретного типу даних цифрової ЕОМ подається скінченна (для цілих) кількість чисел і наближене значення для дійсних (знову пов'язане з *принциповою* скінченністю множини значень).

Для розглянутого прикладу діапазон стандартного цілочислового типу (4 байти) складається з 2^{32} елементів і подає значення від – 2 147 483 648 до 2 147 483 647.

Отже, для цілочислових типів даних можна задавати домен за допомогою повного переліку усіх елементів. Оскільки інші типи в остаточному підсумку (на нижньому рівні абстракції) зводяться до числових, має сенс задавати *скінченний* (!) простір станів для екземплярів класів. Тобто принципово можливо формалізувати інші подробиці ООПар через опис простору станів.

В розглянутому класі Point присутні три операції (методи):

- конструктор (Point() – творець екземплярів, який задає первісний їх стан, тобто ініціалізує їх);
- операції доступу (акцесори) – для запису використовується модифікатор (встановлення стану – SetValue); для читання – селектор (запит поточного стану – GetValue).

Формалізуємо операційну сторону класу (його поведінку) за допомогою опису зміни станів. Для конструктора (створення екземпляра) і для деструктора (знищення екземпляру)

необхідно мати особливий, загальній для усіх класів стан «небуття». У програмуванні використовують спеціальне позначення null – «ніщо».

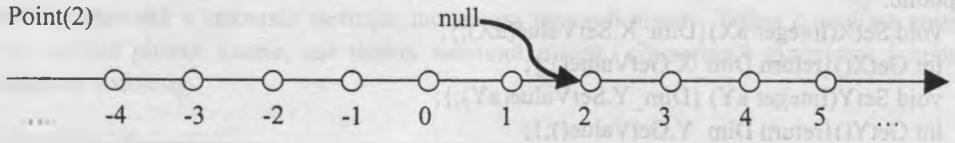

Рис 3. *Опис конструктора за допомогою переходу екземпляра зі стану «небуття» у стан «2».*

SetValue(-1) //при умові, що попередній стан був «2»

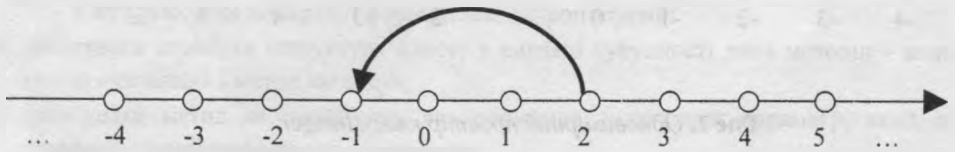

Рис 4. *Опис модифікатора за допомогою переходу екземпляау зі стану «2» у стан «-1».*

GetValue() // при умові, що попередній стан був «2»

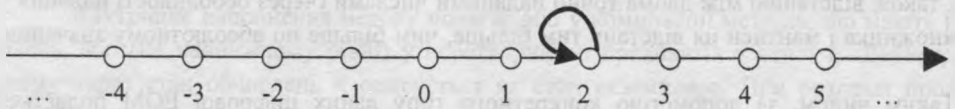

Рис 5. *Опис селектора за допомогою переходу екземпляра зі стану «2» у стан «2» (тобто стан не змінюється).*

Знання про те, як саме операції переводять екземпляр класу з одного стану до іншого, можна формалізувати за допомогою перетворювача предикатів.

Починаючи з методів нижніх рівнів, послідовно застосовуючи розглянуті перетворювачі предикатів, будемо описувати семантику кожного методу за допомогою знаходження післяумови і найслабкішої передумови.

Наприклад, операції розглянутого класу Integer можуть бути формально описані так.

Для конструктора зміна початкового стану обчислень буде виражатися появою («нізвідки») екземпляра класу, ініціалізованого або значеннями за замовчуванням, або параметрами конструктора, а частіш й тим й іншим.

$$\{P \text{ or } i1(null)\}\ Integer(a)\ \{P \text{ or } i1(Value = a)\},$$

де P – стан обчислень, який не містить екземпляр класу Integer з ім'ям i1; i1(null) – неініціалізований (ще не існуючий) екземпляр класу Integer; i1(Value=a) – вираз

стану екземпляра i1 за допомогою опису станів усіх атрибутів у структурі класу; a – конкретне значення типа int.

Модифікатор SetValue змінює значення тільки екземпляра, до якого він застосований. Добрий стиль програмування ("Best practices"), а також закон Деметри, наказує обмежити побічний ефект методів або внутрішніми атрибутами, або переданими параметрами. Що, по суті, забороняє використовувати глобальні змінні.

$$\{P \text{ or } i1(Value = b)\}\ SetValue(a)\ \{P \text{ or } i1(Value = a)\},$$

де P – стан обчислень, який не містить екземпляр класу Integer з ім'ям i1; i1(Value = b) – екземпляр класу Integer, що має стан, який виражається значенням b; i1(Value = a) – екземпляр класу Integer, що має стан, який виражається значенням a; a, b – конкретні значення типа int.

Селектори не змінюють стан екземплярів і необхідні тільки у інформаційних цілях.

$$\{P \text{ or } i1(Value = b)\}\ GetValue()\ \{P \text{ or } i1(Value = b)\},$$

де P – стан обчислень, який не містить екземпляр класу Integer з ім'ям i1; i1(Value = b) – екземпляр класу Integer, що має стан, який виражається значенням b (стан не змінюється); b – конкретне значення типа int.

Предикати, що описують після- та передумови, у крайньому випадку задають єдиний стан. Більш типовим, особливо для найслабкіших передумов, є опис деякої підмножини ізоморфних з цієї точки зору станів. Це можливо, коли підмножина станів задовольняє предикат так, що для опис семантики методу не має значення, який саме стан присутній. Тоді предикат, який описує конкретний стан, буде більш складним, ніж предикат для підмножини.

В розглянутому прикладі класу Point, метод, який модифікує атрибут Вимір_X (Dim_Y), не впливає на атрибут Вимір_Y (Dim_Y). Отже, предикати, які формалізують після- і передумови для цього методу є ізоморфними відносно атрибута Вимір_Y, тобто відрізняються від повного опису стану відсутністю стану цього атрибута (частини стану усього класу).

$$\{P \text{ or } p1(Dim\_X(Value = x1), Dim\_Y(Value = y1))\}$$
$$SetX(z)$$
$$\{P \text{ or } p1(Value = z), Dim\_Y(Value = y1))\}$$

де P – стан обчислювань, який не включає екземпляр класу Point з ім'ям p1; p1(Dim_X(Value = x1), Dim_Y(Value = y1)) – початковий стан екземпляра класу Point; p1(Dim_X(Value = z), Dim_Y(Value = y1)) – результуючий стан екземпляра класу Point; x1, y1, z – конкретні значення типа int.

1. Ковалев С.А. Подход к выделению иерархических структур из словаря данных //Вестник ХПИ. В печати. 2. Айвазян С. А., Бежаева З. И., Староверов О. В Классификация многомерных наблюдений. – М.: Статистика, 1974. – 239 с. 3. Ідентифікація класів – пошук Грааля? /С О. Ковальов //Радіоелектроніка та інформатика. 2002. №1. С.119-123 4. Вирт Н. Алгоритмы+структуры данных = программы /Пер. с англ. Д. Б. Подшивалова. – М.: Мир, 1978. – 280 с. 5. Пратт Т. Языки программирования: разработка и реализация /Пер. с англ. Под ред. Ю. М. Баяковского. – М.: Мир, 1979. – 576 с. 6. Лавров С. С. Основные понятия и конструкции языков программирования. – М.. Финансы и статистика, 1982. – 80 с. 7. Дейкстра Э. Дисциплина программирования. – М.: Мир, 1978. – 274 с. 8. Андерсон Р.

Доказательство правильности программ /Пер. с англ. под ред. Д.Б. Подшивалова. – М.: Мир, 1982. – 168 с. 9. Хоор Ч. Э. Р. Непротиворечивые взаимодополняющие теории семантики языков программирования //Семантика языков программирования. Сборник статей. – М.: Мир, 1980. С. 196-221 10. Абрамов С. А. Элементы анализа программ. Частичные функции на множестве состояний. – М.: Наука. Гл. ред. физ.-мат. лит., 1986. – 128 с. 11. Донаху Дж. Взаимодополняющие определения семантики языка программирования //Семантика языков программирования. Сборник статей. – М.: Мир, 1980. С. 222-394 12. Элиенс А. Принципы объектно-ориентированной разработки программ. 2-е издание.: Пер. с англ. – М.: Издательский дом «Вильямс», 2002. – 496 с. 13. Калинин А. Г., Мацкевич И. В. Универсальные языки программирования. Семантический подход. – М.: Радио и связь, 1991. – 400 с. 14. Лавров С. С. Программирование. Математические основы, средства, теория. – СПб.: БХВ-Петербург, 2001. – 320 с. 15. Себеста Р. У. Основные концепции языков программирования. 5-е изд.: Пер. с англ. – М.: Издательский дом «Вильямс», 2001. – 672 с.

УДК 519.6

П. О. Кравець
Національний університет "Львівська політехніка",
кафедра інформаційних систем та мереж

ОПТИМІЗАЦІЯ ВИПАДКОВОГО ПОШУКУ ГЕНЕТИЧНИМ МЕТОДОМ З РОЗПАРАЛЕЛЮВАННЯМ

It is solved a task of dynamic object search in conditions of uncertainty by application of a classical genetic method with a parallelizing. The dependence of an average amount of retrieval pitches on parameters of a genetic method is investigated.

Розв'язано задачу пошуку динамічного об'єкта в умовах невизначеності за допомогою класичного генетичного методу з розпаралелюванням. Досліджено залежність середньої кількості пошукових кроків від параметрів генетичного методу.

Вступ. Практичне застосування інтелектуальних систем ставить високі вимоги до ефективності методів прийняття рішень в умовах невизначеності. Оптимізація прийняття рішень в умовах невизначеності здійснюється за допомогою адаптивних методів, підкласом яких є еволюційні генетичні методи. Побудова генетичних методів базується на властивостях природного відбору (селекції) поточних рішень та застосування генетичних операторів схрещування і мутації. Природний відбір забезпечує закріплення кращих поточних рішень, схрещування яких з великою імовірністю призводить до їх подальшої