Режим доступу: http://www.crisp-dm.org. 6. Kurgan L. A survey of knowledge discovery and data mining process models / Kurgan L., Musilek P. // Knowledge Engineering Review. – 2006. – №21(1). – P. 1–24. 7. Anand S. Decision Support Using Data Mining. Financial Times Pitman Publishers / Anand S., Buchner A. – London, 1998.8. Anand S. A data mining methodology for cross-sales / Anand S., Hughes P., Bell D. // Knowledge Based Systems Journal.– 1998. – №10.– P. 449–461. 9. Cios K. Diagnosing myocardial perfusion from SPECT bull's-eye maps – a knowledge discovery approach / Cios K., Teresinska A., Konieczna S., Potocka J., Sharma S. // IEEE Engineering in Medicine and Biology Magazine: Special issue on Medical Data Mining and Knowledge Discovery.- 2000.- №19(4). – P. 17–25.10. Cios K. Trends in data mining and knowledge discovery / Cios K., Kurgan L. // Pal N.R. Advanced Techniques in Knowledge Discovery and Data Mining / Pal N.R., Jain L.C.- London: Springer Verlag, 2005. – P. 1–26.11. Kurgan L. Mining the Cystic Fibrosis Data / Kurgan L., Cios K., Sontag M., Accurso F. // Zurada J. Next Generation of Data-Mining Applications / Zurada J., Kantardzic M. – : IEEE Press Piscataway, 2005. – P. 415–444. 12. Pal N.R. Advanced Techniques in Knowledge Discovery and Data Mining / Pal N.R., Jain L.C. – London: Springer Verlag, 2005. 13. Shearer C. The CRISP-DM model: the new blueprint for data mining // Journal of Data Warehousing. – 2000. –№5(4). – P. 13–19.

УДК 004.4'232

О. Овсяк
Львівська філія Київського національного університету
культури і мистецтв

МОДЕЛІ РЕКУРСІЇ ТА РЕКУРЕНЦІЇ

Засобами розширеної алгебри алгоритмів описано моделі рекурсії та рекуренції, наведено приклади їхнього використання.

Recursive and reversing models are described by means of expanded algorithms algebra, the using examples are given.

Вступ

Рекурсія і рекуренція відіграють надзвичайно важливу роль у математиці, алгоритмах і програмуванні. Результатом їх використання у математиці та алгоритмах є компактні вирази функцій та алгоритмів, а у програмуванні, крім компактності коду, досягають істотного зменшення кількості виконуваних операцій і, тим самим, зменшення затрат обсягів пам'яті та часу виконання програм. Однак сьогодні ще немає моделі опису рекурсії та рекуренції у формулах алгоритмів, отриманих застосуванням розширеної алгебри алгоритмів. Власне ці питання і розглядаються у статті. Для створення коректної моделі насамперед необхідно проаналізувати моделі рекурсії та рекуренції у математиці і програмуванні.

Терміни "рекурсії" і "рекуренції" у математиці

В енциклопедії математики [1] так означено **рекурсію**: "РЕКУРСИЯ – способ определения функций, являющийся объектом изучения в теории алгоритмов и других разделах математической логики. Это способ давно применяется в арифметике для определения числовых последовательностей (*прогрессии, чисел Фибоначчи* и пр.). Существенную роль играет рекурсия в вычислительной математике (рекурсивные методы). Наконец, в теории множеств часто используется трансфинитная рекурсия. Долгое время термин рекурсия употреблялся математиками, не будучи

точно определенным. Его приблизительный интуитивный смысл можно описать следующим образом. Значение искомой функции *f* в произвольной точке *x* (под точкой подразумевается набор значений аргументов), определяется, вообще говоря, через значения этой же функции в других точках *y*, которые в каком-то смысле "предшествуют". (Само слово "*рекурсия*" означает "*возвращение*"). …"

Там же [1] дано таке означення **рекурентного співвідношення**: "РЕКУРЕНТНОЕ СООТНОШЕНИЕ, рекурсивная формула, – соотношение вида

$$a_{n+p} = F(n, a_n, a_{n+1}, \dots, a_{n+p-1}),$$

которое позволяет вычислять все члены последовательности $a_1, a_2, a_3, \dots$, если заданы ее первые p членов. Примеры рекурсивного соотношения: *1)* $a_{n+1} = q * a_n$ $(q \neq 0)$- *геометрическая прогрессия*, 2) $a_{n+1} = a_n + d$ – *арифметическая прогрессия*, 3) $a_{n+2} = a_{n+1} + a_n$ – *последовательность чисел Фибоначчи. …*"

На підставі означень і прикладів, наведених у цих означеннях, можна стверджувати, що рекурентне співвідношення є частковим випадком рекурсії. Частковість полягає у тому, що це є специфічна математична формула, а власне не завжди спосіб, на якому робиться наголос в інтуїтивному означенні рекурсії, є математичною формулою, яка описує обчислення всіх членів послідовності, якщо задані попередні значення.

Рекурсія і рекуренція у сучасному програмуванні

Однією із найтиповіших сучасних об'єктно - орієнтованих мов програмування є C# [2–4], яка реалізована на платформі Microsoft Visual Studio .NET [5, 6]. Частина програмного коду, яка має специфікатор доступу [2 – 7] (*public, private* та інші), є статичною або динамічною, повертає (не поверає – *void*) значення конкретного типу і має або не має параметри (вхідні, вихідні, референційні і табличні), називається методом або функцією. У програмному коді, написаному мовою C#, метод може вибирати сам себе. Цей процес названо рекуренційним (рекурсивним), а метод, який вибирає сам себе, названо рекурентним (рекурсивним) [7]. Фактично поняття рекурсії і рекуренції ототожнюються, що проілюстровано прикладом обчислення значення факторіалу з таким кодом програми ([7], с. 291):

```
using System;
class Factorial
    {
    public int  factR (int n)  //То є рекурсивний метод
        {
         int result;
        if( n == 1 )
        return 1;
       result = factR( n – 1 ) * n;  // Рекурсивний вибір методу factR( )
      return  result;
     }
         // А це ітераційний відповідник рекурсивного методу factR( )
   public int  factI( int n )
        {
         int  t, result;
         result = 1;
         for( t = 1, t <= n, t++ )
        result  * = t;
        return  result;
        }
  }
```

```
class Recursion
    {
    public static void  Main( )
         {
          Factorial  f =  new Factorial;
          Console.WriteLine("Рекурсивний метод обчислення факторіалу");
          Console.WriteLine("Факторіал 3 є " + f. factR(3 ));
         Console.WriteLine("Факторіал 4 є " + f. factR(4 ));
         Console.WriteLine("Факторіал 5 є " + f. factR(5 ));

         Console.WriteLine("Для 3 чисел " + f. factI(3 ));
         Console.WriteLine("Для 4 чисел " + f. factI(4 ));
         Console.WriteLine("Для 5 чисел " + f. factI(5 ));
        }
   }
```

Програма має два класи. Це клас з назвою *Factorial*, у якому описано означення методів *factR( )*, який вибирає сам себе для обчислення значення факторіалу (*result = factR(n – 1) * n;*) і *factI( )* для опису обчислення факторіалу з використанням циклу (*for(t = 1, t <= n, t++)*).

Клас *Recursion* використовується для задання початкової точки виконання програми (метод *Main( )*), створення об'єкта класу *Factorial* (*Factorial f = new Factorial;*) і обчислення та виведення на екран комп'ютера, методами *factR( )* та *factI( )*, значень факторіалу для цифр 3 (*Console.WriteLine("Факторіал 3 є " + f. factR(3));* та *Console.WriteLine("Для 3 чисел " + f.factI(3));*), 4 (*Console.WriteLine("Факторіал 4 є " + f.factR(4));* та *Console.WriteLine("Для 4 чисел " + f. factI(4));*), 5 (*Console.WriteLine("Факторіал 5 є " + f. factR(5));* та *Console.WriteLine("Для 5 чисел " + f. factI(5*)).

Використання рекурсії у комп'ютерній системі генерування коду

Комп'ютерна система генерування коду призначена для набору і редагування формул алгоритмів [8 – 12] та автоматичного генерування програмного коду з формул алгоритмів. Головне вікно генератора коду наведено на рисунку. Воно утворене робочим полем набору і редагування формул алгоритмів, лінійкою меню з назвами "Файл", "Операції", "Дії", "Налаштування", "Програма" і "Допомога", а також полем із піктограмами знаків операцій розширеної алгебри алгоритмів (секвентування, елімінування, паралелення, циклічного секвентування, циклічного елімінування і циклічного паралелення), дій (заміни, знищення, задання унітермів, з базою алгоритмів), полів вибору шрифта і кеглю та кнопки.

Для рисування знаків операцій алгебри алгоритмів в абстрактному класі **Term**

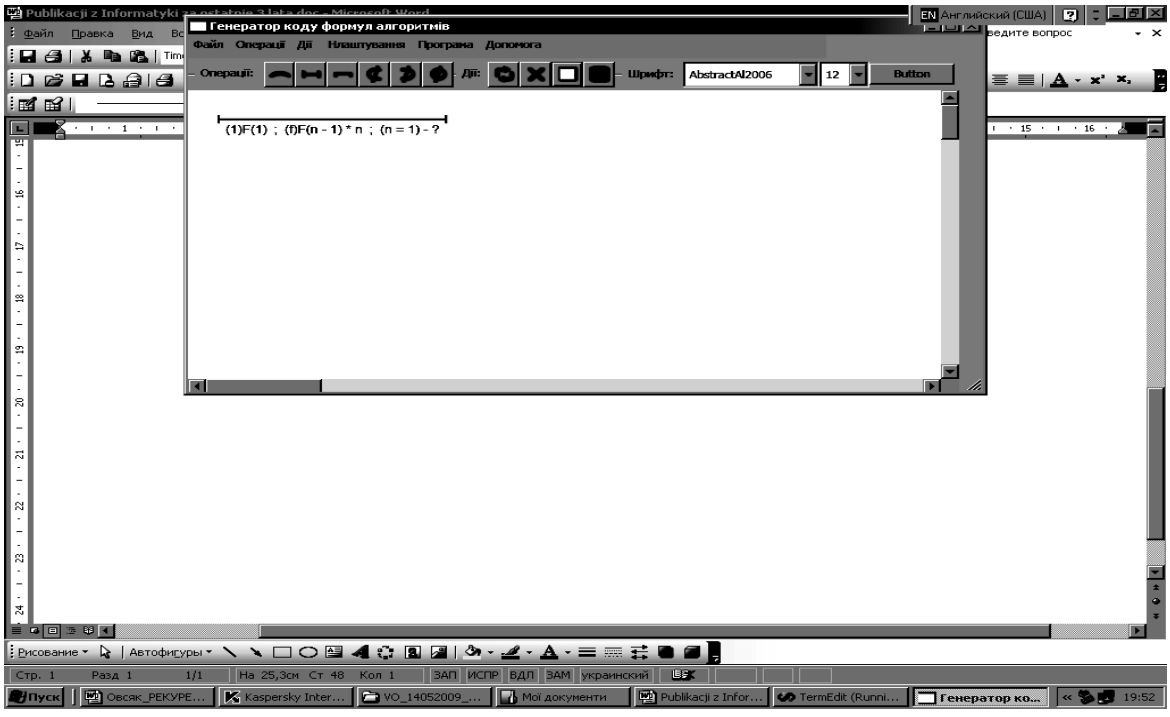
```
public abstract class Term
 {
. . .
public abstract void Draw(MainForm mf, Size f, Brush bb, Brush gb, Pen sp,
          Pen dp,double x, double y, double marginX, double marginY);
       . . .
        }
```

введено абстрактний метод **Draw**( ), вхідними параметрами якого є параметр **mf** (типу головного вікна **MainForm**), двовимірного типу **Size** параметр **f**, два параметри (**bb** і **gb**) типу пензля (**Brush**), два параметри (**sp i dp**) типу пера (**Pen**), чотири змінні (**x, y, marginX, marginY**) типу **double**.

Усі унітерми компютерної системи генерування коду є графічними об'єктами. Тому утворений клас призначений для рисування унітермів. Він названий **Uniterm** і наслідує (**:**)

абстрактний клас **Term**. У класі описано модифікацію (**override**) абстрактного методу **Draw( )** абстрактного класу **Term**. Фрагмент коду класу **Uniterm** з частиною коду методу **Draw( )** є таким

```
public class Uniterm : Term
 {
  . . .
public override void Draw(MainForm mf, Size f, Brush bb, Brush gb, Pen sp,
            Pen dp, double x, double y, double marginX, double marginY)
      {
       . . .
           DrawingVisual poroznijUniterm = new DrawingVisual();
           using (DrawingContext g = poroznijUniterm.RenderOpen())
           {
              g.DrawRoundedRectangle(this.Fill, null, new  Rect(x, y,
                                width, height), marginX, marginY);
              mf.canwasDraw.AddVisual(poroznijUniterm);
           }
       . . .
     }
 . . .
}
```



Головне вікно генератора коду

Метод описує створення об'єкта **poroznijUniterm** системного класу візуалізації графічних об'єктів **DrawingVisual**. Для рисування у ньому використовується метод **RenderOpen()**, який повертає **DrawingContext**, використовуваний для творення графічного вмістимого. Безпосередньо рисування виконується методами **DrawRoundedRectangle(),** з можливістю задання заповнення (**this.Fill –** повне заповнення фігури), контуру (**null** – відсутність контуру) і точки радіуса заукруглення (**marginX, marginY**) та **Rect() –** рисування прямокутника для заданих координат верхнього лівого рогу (**x, y**) й розмірів (**width, height**).

Опис рисування знаку операції секвентування знаходиться у класі **Sequence**, який наслідує (**:**) абстрактний клас **Term**. У класі введено дві змінних **termA** і **termB** типу

```
public class Sequence : Term
   {
      public Term termA;
      public Term termB;
. . .
public override void Draw(MainForm mf, Size f, Brush bb, Brush gb, Pen sp,
            Pen dp, double x, double y, double marginX, double marginY)
      {
       RectangleGeometry rg = new RectangleGeometry();
          . . .

       if (termA != null)
         {
          termA.Draw(mf, f, bb, gb, sp, dp, x, y, marginX, marginY);
          x += termA.width;
         }

       x += (f.Height / 2);
            . . .

     if (termB != null)
         {
          termB.Draw(mf, f, bb, gb, sp, dp, x, y, marginX, marginY);
     }
    . . .
  }
. . .
}
```

абстрактного класу **Term** і переозначення (**override**) абстрактного методу **Draw**.

У методі описано створення об’єкта системного класу **RectangleGeometry**

RectangleGeometry rg = new RectangleGeometry();

Цей системний клас має стандартні методи, які використовуються для рисування частини еліпса – знаку операції секвентування.

Для опису вкладень операцій секвентування в операціїї секвентування на позиціях унітермів використовується рекурсія. Операція секвентування виконується над двома унітермами. Їхня ідентифікація відбувається з використанням змінних **termA** і **termB**. Опис вкладень за змінною **termA** є рекурсивним і має такий вигляд

termA.Draw(mf, f, bb, gb, sp, dp, x, y, marginX, marginY);

Використання рекурсії за змінною **termB** є аналогічним і описано так

termB.Draw(mf, f, bb, gb, sp, dp, x, y, marginX, marginY);

Виконання цих двох рядків починається з аналізу змінних **termA** і **termB**. Значеннями цих змінних є **Uniterm** і **Sequence**. Коли значенням є **Uniterm**, то має місце рекурсія до методу **Draw()**

класу **Uniterm**. Якщо ж значенням є **Sequence**, то відбувається рекурсія теж до методу **Draw(),** але вже класу **Sequence**.

У класах **Elimination : Term** та **Parallelisation : Term**, які теж наслідують (**:**) абстрактний клас **Term**, аналогічно, як і у класі **Sequence**, використовується рекурсія метода **Draw()** для рисування знаків операцій елімінування і паралелення.

Модель опису рекурсії засобами розширеної алгебри алгоритмів

Створимо формулу алгоритму [8 – 12] для обчислення факторіала. Нехай потрібно обчислити значення факторіала, яке позначимо літерою *f*, для *n* чисел, а формулу алгоритму обчислення факторіала позначимо літерою *F* із записаною ліворуч перед нею, у круглих дужках, значення алгоритму *f*, а праворуч за нею, у круглих дужках, кількістю чисел *n*. Отримуємо таке позначення алгоритму*: (f)F(n)*. Відомо, що при $n = 1$ значення факторіала дорівнює *1*. Це з використанням введених позначень запишеться так *(1)F(1)*. Коли ж $n > 1$, то значення факторіала обчислюється за виразом *(f)F(n – 1) * n*. Застосувавши операцію елімінування [8 – 12] для опису обчислення значення факторіала у цих двох випадках, перший з яких отримують при виконанні умови *(n =1)*, а другий – при її невиконанні, матимемо таку формулу:

$$\vdash\!\overline{\qquad\qquad}\!\dashv$$
$$(1)F(1);\ (f)F(n-1) * n;\ (n=1)\text{-}?.$$

Але вище введено позначення формули для обчислення факторіалу, ним є запис *(f)F(n)*. Між ним і отриманою операцією елімінування можна поставити знак рівності. Тому остаточно отримуємо такий вираз:

$$(f)F(n) = \vdash\!\overline{\qquad\qquad}\!\dashv (1)F(1);\ (f)F(n-1) * n;\ (n=1)\text{-}?\ ,$$

який і є формулою рекурсивного алгоритму обчислення значення факторіала.

Якщо допустити, що завжди $n > 1$, то формула обчислення факторіала матиме такий вигляд

$$(f)F(n) = (f)F(n-1) * n.$$

Узагальнена модель опису будь-якої формули рекурсивного алгоритму є такою. Ввести позначення рекурсивного алгоритму. Воно утворене трьома послідовно розташованими складовими. Першу, розташовану зліва, складову утворюють записані у круглих дужках вихідні параметри, назви яких розділені комами. Назва формули алгоритму утворює другу складову, яка записується після вихідних параметрів формули алгоритму. Третя складова утворюється записаними у круглих дужках і розділеними комами вхідними параметрами. Після позначення поставити знак рівності і за ним записати формулу алгоритму. Унітерм, яким є назва алгоритму, записується у потрібному місці у формулі алгоритму.

Якщо формула рекурсивного алгоритму не має вихідних параметрів, то перша складова назви алгоритму опускається. Коли відсутні вхідні параметри, то третя складова утворюється відкритою і закритою круглими дужками.

Висновки

1. Показано, що використовувана у математиці інтуїтивна дефініція рекурсії є більш загальною від означення рекурентного співвідношення, яке отримується із рекурсії.

2. В мові об’єктно-орієнтованого програмування C# поняття рекурсії і рекуренції є тотожніми, що не відповідає інтуїтивним означенням рекурсії та рекуренції, оскільки не завжди рекурсія є рекурентним співвідношенням.

3. Розроблено модель побудови формули рекурсивного алгоритму, яку можна застосувати для створення будь-якої формули рекурсивного алгоритму.

1. Математическая энциклопедия. – М.: “Советская энциклопедия”, 1984. – Т 4. 1216 с.
2. Нэш Трей. C# 2008: ускоренный курс для профессионалов. – М.: “ИД Вильямс”, 2008. – 576 с.
3. Мак-Дональд Мэтью. WPF. Windows presentation foundation .NET 3.5 с примерами на C# 2008.

Для профессионалов. – М., СПб., К.: "ИД Вильямс", 2008. – 928 с. 4. Petzold Charles.Programirowanie Microsoft WINDOWS w języku C#. – Warszawa: „RM", 2003. – 1161 s. 5. Powers Lars, Snell Mike. Microsoft Visual Studio – 2005. Księga eksperta. – Gliwice: „Helion", 2007. – 840 s. 6. Троэлсен Эндрю. Язык программирования C# и платформа .NET 2.0. – – М., СПб., К.: "ИД Вильямс", 2007. – 1168 с. 7. Schildt Herbert. C#. Kurs podstawowy. – Kraków:"Edittion 2000", 2002. – 638 с. 8. Овсяк В. Засоби еквівалентних перетворень алгоритмів / Овсяк В. // Доповіді національної академії наук України. – 1996. – №9. – С.83–89. 9. Овсяк В. АЛГОРИТМИ: аналіз методів, алгебра впорядкувань, моделі, моделювання / В. Овсяк. – Львів, 1996. – 132 с. 10. Овсяк В. АЛГОРИТМИ: методи побудови, оптимізації, дослідження вірогідності / В. Овсяк. – Львів: Світ, 2001. – 160 с. 11. Owsiak W., Owsiak A., Owsiak J. Teoria algorytmów abstrakcyjnych i modelowanie matematyczne systemów informacyjnych / Owsiak W., Owsiak A., Owsiak J. – Opole: Politechnika Opolska, 2005. – 275 s. 12. Ovsyak V.K. Computation models and algebra of algorithms / V.K. Ovsyak // Інформаційні системи та мережі: Вісник Нац. ун-ту "Львівська політехніка". – 2008. – № 621. – С.3 – 18.

УДК 004.22(0.23)

В. Лахно, А. Петров
Луганський національний аграрний університет,
кафедра економічної кібернетики

МОДЕЛЮВАННЯ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ КОРПОРАТИВНИХ СИСТЕМ ПІДПРИЄМСТВ З ВИКОРИСТАННЯМ ТЕОРІЇ ІГОР І МАРКІВСЬКИХ ПРОЦЕСІВ

Розглянуто питання моделювання системи захисту інформації, побудованої з використанням теорії ігор і теорії випадкових марківських процесів, за допомогою якої розглядається сукупність проектів систем захисту інформації, розраховуються вірогідність здійснення загроз для ресурсів корпоративних інформаційних систем і ризики за кожною загрозою, і на основі цих показників вибирається оптимальний проект.

The article deals with the issue of information security modeling system built using game theory and the theory of random Markov process, whereby a set of projects considered information security systems, calculate the probability of threats to corporate information systems resources and risks of each threat, and based on these indicators selected the best project.

Постановка проблеми

Найціннішою в корпоративних мережах суб'єктів господарської діяльності є інформація, яка становить інтерес для конкурентів. Про серйозність проблеми свідчить хоча б такий факт, що одна людина, що має доступ до серверу корпоративної мережі або бази даних, за незначний час може повністю паралізувати діяльність будь-якої компанії. Для цього достатньо ввести в програмне забезпечення системи всього декілька десятків рядків коду програми-вірусу. Якщо система не матиме спеціальних засобів захисту, то це загрожує як мінімум величезними економічними втратами.

Аналіз попередніх досліджень

У більшості вітчизняних і зарубіжних джерел вважають головною загрозою скоординовані напади хакерів на державні, корпоративні і приватні мережі одночасно з багатьох точок земної кулі. Такого роду напади характеризуються одночасним надсиланням мільйонів пакетів, великою