УДК 519.681.2
DOI https://doi.org/10.32689/maup.it.2025.1.22

Олексій ПІСКУНОВ
кандидат фізико-математичних наук, старший науковий співробітник, доцент кафедри прикладної математики, Державний університет «Київський авіаційний інститут», oleksii.piskunov@npp.nau.edu.ua
***ORCID:** 0000-0002-9200-3422*

Наталя ТУПКО
кандидат фізико-математичних наук, доцент, доцент кафедри прикладної математики, Державний університет «Київський авіаційний інститут», natalia.tupko@npp.nau.edu.ua
***ORCID:** 0000-0003-0625-3271*

Олександр ВАСИЛЬЄВ
кандидат фізико-математичних наук, доцент, доцент кафедри оптимального керування і економічної кібернетики, Одеський національний університет імені І.І. Мечникова, av5111955@gmail.com
***ORCID:** 0000-0002-3826-4883*

Надія ТОПІХА
здобувач магістратури кафедри прикладної математики, Державний університет «Київський авіаційний інститут», 6856645@stud.nau.edu.ua
***ORCID:** 0009-0005-1654-7179*

ФОРМАЛЬНЕ ТЕСТУВАННЯ ФУНКЦІЙ ДІЙСНОГО АРГУМЕНТУ

Анотація. *У статті розглянуто алгебраїчний підхід до проєктування та тестування програмного забезпечення.*

Метою статті *є застосування методів формального проєктування програмного забезпечення (ПЗ) на прикладі розробки функцій підрахунку довжини дуги меридіану. Методи формальної розробки ПЗ досі не набули широкого використання в практичній діяльності, але враховуючи формальну природу мов програмування, їх впровадження в математичні дослідження виглядає цілком обґрунтованим.*

Методи дослідження: *під час дослідження використовуються базові положення методу формальної розробки RAISE, які дозволяють застосовувати формальну логіку. Зокрема, розглядається використання інструментів розробки програмного забезпечення від RAISE Method Group для специфікації функцій обчислення довжини дуги меридіана та визначення умов створення драйвера тестів для цих функцій на ранньому етапі программування. Специфікація умов для тестування формулюється у вигляді аксіом з використанням абстрактного аплікативного підходу.*

Наукова новизна *дослідження полягає в тому, що мова формальних специфікацій RAISE Specification Language (RSL) та спеціалізована утиліта є достатньо зручними інструментами для повсякденної роботи маленьких груп наукових співробітників, які не мають спеціальних навичок в галузі формальної разработки ПЗ. Особливо цікавила область математичних досліджень, зокрема для розробки складних функцій дійсного аргументу. Окрім основної мети, було уточнено сім коефіцієнтів (C0, C2, C4, C6, D2, D4, D6) розкладання розглянутих функцій у ряд Маклорена, складено формули та виконано розрахунки для наступних двох коефіцієнтів (C8, D8). Запропоновано практичний прийом розробки узагальнених драйверів тесту, заснований на успадкування класу, що розробляється від муляжу (mock class) з використанням можливостей. NET, таких як класи, що розділяються (partial class). Загалом стаття спрямована на зменшення розриву між теоретичними перевагами методів формальної розробки та їх недостатньо широким застосуванням у практиці.*

Висновки: *алгебраїчне проєктування та тестування базується на математичних принципах, що дозволяє: уникати двозначності і неоднозначності в описі функціональності; забезпечувати точність та однозначність у формулюванні вимог до програми; автоматизувати процес генерації тестових випадків та перевірки роботи, відповідно підвищувати надійність; виявляти і усувати помилки ще на стадії розробки та прискорити розробку ПЗ.*

Ключові слова: *алгебраїчне проєктування тестів, RAISE Specification Language, функція дійсного аргументу.*

Oleksii PISKUNOV, Natalia TUPKO, Alexander VASILIEV, Nadiia TOPIKHA. FORMAL TESTING OF REAL-VALUED FUNCTIONS

Abstract. *The article discusses the algebraic approach to software design and testing.*

The aim *of the study is to apply methods of formal software design to the development of functions for calculating the length of a meridian arc. Although formal software development methods have not yet been widely adopted in practical activities, their application in mathematical research seems reasonable due to the formal nature of programming languages.*

Research methods: *the research utilizes the basic principles of the RAISE formal development method, which allows the use of formal logic. Specifically, the study explores the use of software development tools provided by the RAISE Method Group to specify functions for calculating the length of a meridian arc and to define conditions for creating a test driver for these functions at an early programming stage. The testing conditions are specified in the form of axioms using the abstract applicative approach.*

Scientific novelty *the novelty of the study lies in demonstrating that the RAISE Specification Language (RSL) and specialized utilities are sufficiently convenient tools for small research teams that lack specialized skills in formal software development. The focus is on mathematical research, particularly on developing complex real-valued functions. In addition to the primary goal, seven coefficients (C0, C2, C4, C6, D2, D4, D6) of the Maclaurin series expansion for the considered functions were clarified. Formulas and calculations for the next two coefficients (C8, D8) were also performed. A practical method for developing generalized test drivers based on inheriting a class from a mock class was proposed, utilizing. NET features such as partial classes. Overall, the article aims to reduce the gap between the theoretical advantages of formal development methods and their insufficient practical application.*

Conclusions: *algebraic design and testing are based on mathematical principles, enabling the following: avoiding ambiguity in functionality descriptions; ensuring precision and clarity in software requirements formulation; automating the generation of test cases and verification processes, thereby enhancing reliability; identifying and fixing errors at the development stage; and accelerating software development.*

Key words: *algebraic test design, RAISE Specification Language, real-valued function.*

Постановка проблеми та аналіз останніх досліджень і публікацій. У роботі вирішується задача застосування методів формального тестування в практиці програмування, а саме пропонуються засоби формулювання умов для створення димових тестів для функцій дійсного аргументу на прикладі обчислення довжину дуги меридіана.

Зазначимо, що цей підхід використовувався в роботах і методах розробки програмного забезпечення багатьох авторів. Зокрема, у Б. Мейера в проєктуванні за контрактом [1], у методі формальної розробки RAISE [2], у статті щодо формалізації принципу підстановки Б. Лісков [3], у видатній монографії Коді-Вейта [4])та інших. Загалом, цей підхід, імовірно, бере початок від роботи Грассмана з формалізації арифметики 1861 року [5]. Формалізація вимог до димових тестів записується у вигляді аксіом у схемах мови формальних специфікацій RSL – RAISE Specification Language. Результати, представлені в доповіді, продовжують зусилля, розпочаті в [6, 7, 8]. Цю роботу можна розглядати як виконання ще одного невеликого кроку (як це пропонував робити Д.Л. Парнас у своєму огляді [9]) по застосуванню методів формальної розробки в практиці програмування.

Виклад основного матеріалу. У роботі проаналізовано алгебраїчний підхід до проєктування та тестування функції дійсного аргументу. Враховуючи, що багато звичних функцій дійсного аргументу (наприклад, *sqrt, log*) розглядалися в монографії Коди-Вейта [4], в якості досліджуваних функцій було обрано пару корисних і достатньо простих функцій обчислення довжини дуги меридіана. Наступна схема мови RSL задає необхідні величини, типи та сигнатуру обох функцій. Функція *len* за заданою широтою в радіанах повертає довжину дуги меридіана в метрах. Функція *latitude* за заданою довжиною дуги меридіана в метрах від екватора повертає широту в радіанах (лістинг 1):

```
scheme values = class
 value
    sin        : Real -> Real,
    Pi         : Real,                    -- 3.1415926
    a          : Real = 6378137.0, --   екв.піввісь, WGS-84
    maxLen : Real = len (maxLatitide),
    maxLatitide : Real
 type
    Radian  = {|B: Real :- B >= 0.0 /\ B <= 90.0 * (Pi / 180.0)|},
     Meter   = {|X: Real :- X >= 0.0 /\ X <= maxLen|}
 value
      len   : Radian  ->  Meter,
      latitude   : Meter   ->  Radian
end
```

Тепер на мові реалізації (в даному випадку це C#) можна записати сигнатуру методів і заглушки (stub function) для драйвера тестів. До речі, схоже буде зручно перевизначати заглушки для драйвера тестів у похідному класі справжньою реалізацією функцій, що розробляються. Код для заглушок:

```
public abstract class geoA {
   public abstract double len     (double radian);
   public abstract double latitude (double meter);
}
public  class stub: geoA {
```

```
      override public double len     (double radian) {return radian};
      override public double latitude (double meter)  {return meter};
}
public  class geo: stub {
   // місце для розроблюваних функцій
}
```

Таким чином конструкція спадкування об'єктно-орієнтованої мови використовується для створення поліморфного (узагальненого) коду самого драйвера. Відповідно драйвер тестів не вимагатиме переробки у процесі кодування необхідних функцій.

Реальна перевірка коректності обчислень функцій на місцевості неможлива без спеціального геодезичного обладнання та відповідної освіти. Проте з алгебраїчної точки зору зрозуміло, що попарне застосування функцій має давати результат, який точно збігається з початковою величиною. Ця умова дозволяє негайно записати вимоги для димових тестів. А за допомогою функцій-заглушок можна на самому ранньому етапі розпочати розробку драйвера тестів:

```
values
scheme test = extend values with class
  axiom
  [smoke_test_1]
      all X : Meter  :- len(latitude( X )) - X is 0.0,
  [smoke_test_2]
      all B : Radian :- latitude(len( B )) - B is 0.0
end
```

Звісно, в жодній реалізації обидві аксіоми не виконуватимуться повністю. Порушення аксіом зазвичай виникають через помилки округлення [4], які призводять до накопичення певного відхилення від нуля під час обчислень. Таким чином, за допомогою драйвера тестів для згаданих аксіом можна отримати два набори пар (або дві функції): (метри, відхилення) в одному наборі та (радіани, відхилення) в іншому наборі для подальшого аналізу.

Повернемося до досліджуваних функцій: по-перше, функція *len* виводиться із рівності [10, с. 50]:

$$X = \int_0^B MdB = a\left(1-e^2\right)\int_0^B \left(1-e^2\sin^2 B\right)^{-\frac{3}{2}} dB.$$

Для цього підінтегральна функція розкладається в ряд Маклорена та приймає наступний вигляд після інтегрування:

$$X = a\left(1-e^2\right)\left(C_0 B - C_2\sin 2B + C_4\sin 4B - C_6\sin 6B + C_8\sin 8B - \ldots\right), \quad (1)$$

де

$$C_0 = 1 + \frac{3e^2}{4} + \frac{45e^4}{64} + \frac{175e^6}{256} + \frac{11025e^8}{16384} + \ldots;$$

$$C_2 = \frac{3e^2}{8} + \frac{15e^4}{32} + \frac{525e^6}{1024} + \frac{2205e^8}{4096} + \ldots;$$

$$C_4 = \frac{15e^4}{256} + \frac{105e^6}{1024} + \frac{2205e^8}{16384} + \ldots;$$

$$C_6 = \frac{35e^6}{3072} + \frac{315e^8}{12288} + \ldots;$$

$$C_8 = \frac{315e^8}{131072} + \ldots.$$

Зазначимо, що наведені вище коефіцієнти містять більше членів, ніж у [10, с. 50], та уточнені в результаті проведених обчислень і відрізняються від тих, що приведені у [11, с. 28]. Для порівняння використано точнішу специфікацію рівняння з п'ятьма додатковими членами та вихідну версію [10, с. 50].

На мові RSL нескладно записати самі вимоги до вибраних функцій для їх реалізації (вихідна версія, листинг 2):

```
values
scheme length1 =  extend values with class
   value
     C0  : Real  =  6367449.1458,
     C2  : Real  =  16038.5086,
     C4  : Real  =  16.8326,
     C6  : Real  =  0.0220
   axiom
      all  B : Radian :-
         len(B) is C0 * B - C2 * sin(2.0 * B)
                  + C4 * sin(4.0 * B) - C6 * sin(6.0 * B)
end
```

По-друге, функція *latitude* виводиться із рівності [10, формула (3.21)

$$B = \beta + D_2 \sin 2\beta + D_4 \sin 4\beta + D_6 \sin 6\beta + D_8 \sin 8\beta + ..., \quad (2)$$

яка отримана за допомогою обертання тригонометричних рядів рівності [10, с. 51,52)]. Причому:

$$\beta = \frac{X}{C_0};$$

$$D_2 = \frac{C_2}{C_0}\left(1 + \frac{C_4}{C_0} - \frac{C_2^2}{2C_0^2}\right);$$

$$D_4 = \frac{C_2^2}{C_0^2} - \frac{C_4}{C_0};$$

$$D_6 = \frac{C_6}{C_0} - \frac{3C_2}{C_0}\left(\frac{C_4}{C_0} - \frac{C_2^2}{2C_0^2}\right);$$

$$D_8 = \frac{C_8}{C_0} - \frac{4C_6}{C_0}\left(\frac{C_4}{C_0} - \frac{C_2^2}{2C_0^2}\right).$$

Як і вище, для порівняння використано дві версії функції. Наступна подана специфікація є записом формули [10, с. 51]:

```
length1
scheme latitude1 = extend length1 with   class
  value
     e    : Real  =   10.0,
     D2 : Real  =   251882658.8 * e ** -10.0,
     D4 : Real  =   37009.6     * e ** -10.0,
     D6 : Real  =   74.5        * e ** -10.0
  axiom
    all X : Meter :-
       latitude( X ) is
         let beta = X / C0
           in beta + D2 * sin(2.0 * beta)
                 + D4 * sin(4.0 * beta) + D6 * sin(6.0 * beta)
            end
end
```

В цій специфікації відсутній ще один порахований доданок з коефіцієнтом *D8.* Результати роботи реалізації цих функцій мовою C# з коефіцієнтами (*C0, C2, C4, C6, D2, D4, D6*), значення яких розраховуємо за допомогою зазначених схем, на рис. 1 і 2 відповідають кривим із позначенням *book coefs*.

Таким чином на ранніх етапах V-подібної розробки програмного забезпечення вдалося сформулювати формальні вимоги до самого програмного забезпечення та драйвера тестів.

Для тестування було використано по двадцять тисяч випадкових значень на інтервалі від 0 до 150 градусів. Зазначимо, що розглядалось 150 градусів, а не 90, як було записано в специфікації лістингу 1, для переконання в періодичності відхилень від ідеального значення 0. Обидва тести, які вираховують відхилення від 0 в залежності від координати (в метрах або радіанах), показали, що вони мають явно виражену періодичну природу (рис. 1 та рис. 2). Як зазначалося вище, одну криву на рисунках отримано з використанням коефіцієнтів формул, взятих з [10, с. 50] та [10, с. 51]. Другу криву – було отримано з використанням додатково порахованих коефіцієнтів із збільшеною кількістю доданків, формули (1) та (2).

Результати роботи реалізації функцій із явно порахованими коефіцієнтами на рис. 1 та рис. 2 (відповідають кривим із позначенням *calculated coefs*), показали поліпшення результатів тестування. Відносна помилка для першого димового тесту складає 1.16•10−9 відсотка (9.6•10−5), −6.7•10−8 відсотка (8.6•10−10 радіана в абсолютних значеннях).

Розглянемо схему RSL з явним підрахунком коефіцієнтів для першого тесту (доданий п'ятий коефіцієнт *C8*):

```
values
scheme length5 =  extend values with class
   value
      e2  : Real  =  0.006694379990,    -- кв.эксентриситета
      C0 : Real = a *(1.0-e2) * (1.0 +    3.0/4.0     * e2
                                  +   45.0/64.0    * e2 ** 2.0
                                  +  175.0/256.0   * e2 ** 3.0
                                 +11025.0/16384.0  * e2 ** 4.0),
      C2 : Real =   a * (1.0 - e2)  * (  3.0/8.0      * e2
                                 +   15.0/32.0     * e2 ** 2.0
                                 +  525.0/1024.0   * e2 ** 3.0
                                 +  2205.0/4096.0  * e2 ** 4.0),
      C4 : Real =   a * (1.0 - e2) * (   15.0/256.0   * e2 ** 2.0
                                 +   105.0/1024.0  * e2 ** 3.0
                                 +  2205.0/16384.0 * e2 ** 4.0),
      C6 : Real =   a * (1.0 - e2)* (    35.0/3072.0  * e2 ** 3.0
                                  + 315.0/12288.0 * e2 ** 4.0),
      C8 : Real =   a * (1.0 - e2)* (   315.0/131072.0* e2 ** 4.0)
  axiom         -- -- Colin Maclaurin, 1698-1746
       all  B : Radian :-
           len(B) is ( C0 * B - C2 * sin(2.0 * B) + C4 * sin(4.0 * B)
                           - C6 * sin(6.0 * B) + C8 * sin(8.0 * B))
 end
```

Зауважимо, що значення перерахованих коефіцієнтів дещо відрізняються від представлених на лістингу 2:

C0=6367449.14576652
C2=16038.5086150243
C4=16.8325996501986
C6=0.0219825966927238
C8=3.05786850009824E-05

Результати обчислень першого тесту, де вісь абсцис показує відстань від екватора до поточної точки в градусах, вісь ординат показує отримане відхилення від нуля (рис. 1).

Схема з явним підрахунком коефіцієнтів другого тесту (доданий п'ятий коефіцієнт *D8*):

```
length5
scheme latitude5 = extend length5 with   class
   value
     D2 : Real = C2/C0 * (1.0 + C4/C0 -  C2**2.0 / (2.0 * C0**2.0) ) ,
     D4 : Real = C2**2.0/C0** 2.0 - C4/C0,
```

```
      D6 : Real = C6/C0 - 3.0 * C2 / C0 * (C4/C0 - C2**2.0 / (2.0 * C0**2.0)),
      D8 : Real = C8/C0 - 4.0 * C6 / C0 * (C4/C0 - C2**2.0 / (2.0 * C0**2.0))
   axiom
      all X : Meter :-
          latitude( X ) is
             let beta = X/C0
                in beta + D2 * sin(2.0*beta) + D4 * sin(4.0*beta)
                                + D6 * sin(6.0*beta) + D8 * sin(8.0*beta)
             end
end
```

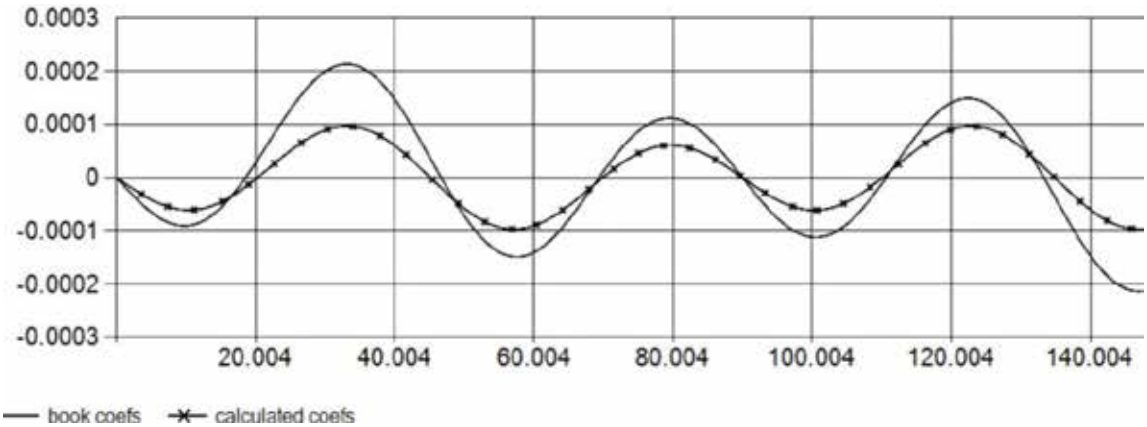

Рис. 1. Перший тест

Значення перерахованих коефіцієнтів наступні:
C0=6367449.14576652
D2=0.00251882657688729
D4=3.70095511127976E-06
D6=7.44751369051007E-09
D8=4.80964571638517E-12

Результати обчислень другого тесту, де вісь абсцис показує відстань від екватора до поточної точки в градусах, вісь ординат показує отримане відхилення від нуля (рис. 2).

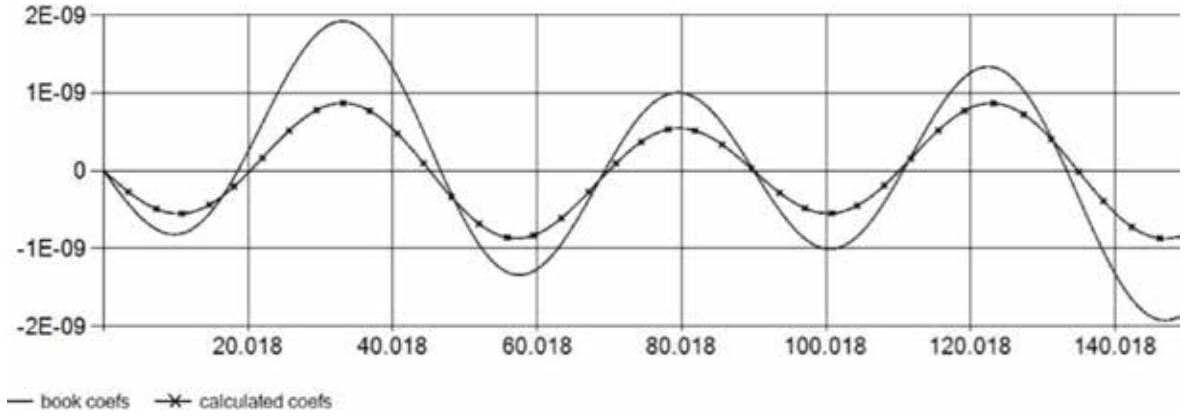

Рис. 2. Другий тест

Аналіз графіків показує, що:

- перераховані коефіцієнти дещо зменшують відхилення від 0 при попарному застосуванні функцій;
- оскільки обидві функції за своєю природою є строго монотонними, формально доцільно виконати тести на перевірку їхньої монотонності на окремих відрізках області визначення. Зокрема, на тих відрізках, де спостерігається найбільше зростання від'ємної похибки між сусідніми точками осі абсцис.

Наприклад, біля точки 0 перевіримо виконання умови монотонності:

```
values
scheme monofunc = extend values with class
    axiom
    [smoke_test_3]
         all B0, B1 : Radian:- B0 < B1 =>
                  len(B0) <= len(B1),
    [smoke_test_4]
         all X0, X1 : Meter :- X0 < X1 =>
                   latitude(X0) <= latitude(X1)
 end
```

За допомогою великої серії тестів (100000000, 25 хвилин обчислень), біля нуля не вдалося виявити порушень монотонності. Частковий код для перевірки цих умов представлений на лістингу 3. Число *double.Epsilon* – це мінімальна додатнє число бібліотеки.Net Framework, яке можна представити в типі *double*. Тобто числа *B1* і *B0* в операторі *B1=B0+ double.Epsilon*; виявляються сусідніми точками на розглянутому відрізку. Результати експерименту показують, що два значення функції *len* на сусідніх точках виявляються або рівними, або значення функції на більшому значенні аргументу виявляється більшим.

Частковий код тесту (лістінг 3):

```
Console.WriteLine("# testNo len0 len1 difference" );
B0 = math.degree2radian(BSta);
for ( int i = 0;  i < nums;  i++, B0=B1+double.Epsilon * steps){
  B1 = B0 + double.Epsilon;
  len0 = arc.len(B0);
  len1 = arc.len(B1);
  diff = len1 - len0;
  if (len0 > len1)
    Console.WriteLine("{0}:[{1},{2}]  :  {3}",i,len0,len1,diff);
}
```

Крім того, можна дослідити причину відхилень у точках локальних екстремумів. Але, ймовірно, ці відхилення є наслідком розкладання в ряд Маклорена, і дослідження цього питання виходить за межі нашої роботи.

На завершення утилітою RSLTC [12] згенеруємо умови впевненості (confidence condition), які допомагають записати тести для визначення коректних та некоректних класів еквівалентності та граничних значень:

```
c:\>rsltc32.exe -cc latitude1.rsl
```

В результаті отримаємо:

```
rsltc version 2.5 of Sun Jan 9 20:23:27 2005 Loading values ...
values.rsl:8:60: CC:
-- application arguments and/or precondition let x = Pi, y =
 180.0 in y ~= 0.0 end values.rsl:8:30: CC:
-- subtype not empty      *** ДРУГА УМОВА*** exists B :
Real :- B >= 0.0 /\ B <= 90.0 * (Pi / 180.0) values.rsl:9:30: CC:
-- subtype not empty      *** ТРЕТЯ УМОВА *** exists X :
Real :- X >= 0.0 /\ X <= maxLen
. . .
rsltc completed: 7 confidence condition(s) generated rsltc completed: 0 error(s) 0
warning(s)
```

Зазначимо, що друга умова:

```
exists B : Real :- B >= 0.0 /\ B <= 90.0 * (Pi / 180.0)
```

вказує на існування трьох класів еквівалентності для функції *len*: значення широти B, які менші за 0; значення у відрізку [0, 90 × (Pi/180)]; значення широти які більше, ніж 90 × (Pi /180).

Третя умова:

```
exists X : Real :- X >= 0.0 /\ X <= maxLen
```

теж вказує на необхідність використання трьох класів еквівалентності для функції *latitude*.

З точки зору програмування більш корисними видаються функції *len* і *latitude*, вже від двох аргументів. Ці функції обчислюють відстані не від екватора до заданої точки на меридіані, а від однієї заданої точки на меридіані до іншої заданої точки на меридіані. Їх можна буде використовувати для формального тестування функцій для вирішення прямої та зворотної геодезичних задач. Тому, насамкінець, для ілюстрації наведемо ще одну схему для їх визначення. На цій схемі функції визначаються за допомогою лямбда-виразу. Крім того, на схемі є умови для алгебраїчного тестування цих функцій від двох аргументів:

```
latitude1
scheme values2 = extend latitude1 with  class
   value
      len: Radian >< Radian ->  Meter =
            -\(B0: Radian, B1: Radian) :-
                let x0 = len(B0), x1 = len(B1)
                   in x1 - x0 end,
       latitude: Radian >< Meter -> Radian =
              -\(B: Radian, X: Meter) :-
                         let bx = len (B)
                             in latitude(bx + X)    end
  axiom
  [smoke_test_5]
       all B: Radian, X: Meter :- len(B, latitude(B,X)) - X is 0.0,
  [smoke_test_6]
          all B0, B1: Radian :- latitude(B0, len(B0,B1)) - B1 is 0.0
end
```

Висновки. У роботі отримані наступні результати:

– наведено завершений приклад практичного застосування методів формальної розробки функцій дійсного аргументу;

– мовою формальних специфікацій RSL були сформульовані алгоритми обчислення чотирьох різних функцій для отримання довжини дуги меридіанів;

– мовою формальних специфікацій RSL були сформульовані умови для шести димових тестів;

– для однієї з функцій (*len*) виконано перевірку на порушення умови монотонності;

– була використана утиліта RSLTC для статичної перевірки текстів специфікації та отримання класів еквівалентності;

– уточнено сім коефіцієнтів (*C0, C2, C4, C6, D2, D4, D6*) для обчислення розглянутих функцій;

– складено формули та виконано розрахунки для наступних коефіцієнтів (*C8, D8*);

– запропоновано технічний прийом для розробки узагальненого коду драйвера тестів;

– наведено закінчений приклад виконання алгебраїчного тестування функцій дійсного аргументу.

Список використаних джерел:

1. Мейєр Б. *Object-oriented software construction* (2-е вид.). Санта-Барбара: ISE Inc, 2000. 1284 с.
2. Haxthausen A. Lecture Notes on The RAISE Development Method. Kongens Lyngby: DTU, 1999. 20 с.
3. Піскунов О. Г. Про відмінності між поняттями типу та класу. *Вісник Київського національного університету імені Тараса Шевченка. Серія «Фізико-математичні науки»*, 2015, № 3, с. 106–114.
4. Cody W., Waite W. Software manual for the elementary functions. New Jersey: Prentice-Hall, 1980. 289 с.
5. Grassmann H. Arithmetik für höhere. Stettin: Verlag von Friedrich Nagel, 1861. 224 с.

6. Піскунов О. Г., Тупко Н. П., Топіха Н. В. Формальний погляд на тестування функцій дійсного аргументу. Сучасні методи, інформаційне, програмне та технічне забезпечення систем керування організаційно-технічними та технологічними комплексами: матеріали XI Міжнародної науково-технічної Internet-конференції, м. Київ, 27 лист. 2024 р. / НУХТ, 2024, с. 218–219.

7. Піскунов О., Тупко Н., Петренко І. Алгебраїчне проєктування програмного забезпечення. *Інформаційні технології та суспільство*, 2024, № 5 (11), с. 50–59.

8. Піскунов О. Г., Сіренко А. Г. Мови формальних специфікацій та документування методів класу. Штучний інтелект та інформаційні технології (AIAT-2024): матеріали першої міжнародної науково-практичної конференції, м. Київ, 3–4 червня 2024 р. / НУХТ, 2024, с. 340–342.

9. Parnas D. L. Really rethinking 'formal methods'. IEEE Computer Society, Computer, 2010, № 43, с. 28–34.

10. Serapinas B. Geodetic Foundations of Maps. URL: https://cutt.ly/8e9clPjJ (дата звернення: 10.01.2025).

11. Palikaris A., Tsoulos L., Paradissis D. New Meridian Arc Formulas for Sailing Calculations in Navigational GIS. URL: https://www.researchgate.net/publication/236611373 (дата звернення: 10.01.2025).

12. George C. RAISE Tool User Guide. UNU-IIST, 2008, 162 p. URL: https://raisetools.github.io/material/documentation/ug.pdf (дата звернення: 10.01.2025).