М.С. Львов

Основи комп’ютерної алгебри та алгебраїчних обчислень

Навчальний посібник для студентів університетів комп’ютерних та математичних спеціальностей

Херсон 2018

Рекомендовано до друку Вченою радою Херсонського державного університету
(протокол № від)

Рецензенти:

Академік НАН України, доктор фізико-математичних наук, професор **Летичевський О.А.**,
Інститут кібернетики НАН України ім. акад. В.М.Глушкова, завідувач відділу теорії цифрових автоматів.

доктор фізико-математичних наук, професор **Нікітченко М. С.**,
Києвський національний університет ім. Тараса Шевченко, професор кафедри теорії і технології програмування факультету кібернетики

М.С. Львов.

Основи комп'ютерної алгебри та алгебраїчних обчислень. Навчальний посібник

Анотація

Ця книга є навчальним посібником з основ комп'ютерної алгебри та алгебраїчного програмування. Вона містить навчальний матеріал з основ комп'ютерної арифметики (елементарної теорії чисел), технологій алгебраїчного програмування та системи алгебраїчного програмування АПС, методології програмування багатосортних алгебраїчних систем (алгебраїчних обчислень), основних алгоритмів комп'ютерної алгебри та прикладів застосування цих алгоритмів до розв'язання прикладних задач комп'ютерної алгебри.

Викладення основ алгебраїчного програмування використовує систему алгебраїчного програмування АПС, розроблену в інституті кібернетики НАН України під керівництвом акад. О.А. Летичевського.

Оригінальною є методологія програмування алгебраїчних обчислень, основана на поняттях конструктивного розширення, наслідування та морфізмів багатосортних алгебр.

Главу 4 присвячено методам комп'ютерної тригонометрії. Глава 5 містить традиційне викладення класичних алгоритмів комп'ютерної алгебри. Це алгоритми множення та факторизації поліномів, конструктивна теорія поліноміальних ідеалів (базиси Гребнера), алгоритм Штурма уточнення дійсних коренів поліномів, а також приклади використання цих алгоритмів. У заключному параграфі 5-ої викладено оригінальний метод розв'язання систем лінійних нерівностей, оснований на побудові канонічних форм представлення полігонів – розв'язків систем цих систем.

За традицією, останню, шосту главу присвячено науковим інтересам автора: алгоритмам комп'ютерної алгебри розв'язання задач статичного аналізу комп'ютерних програм.

Книга буде корисною студентам, магістрантам та аспірантам комп'ютерних та математичних спеціальностей університетів, що мають вивчати та використовувати методи комп'ютерної алгебри та технології алгебраїчного програмування.

Основи комп’ютерної алгебри та алгебраїчних обчислень

Вступ

Серед прикладного програмного забезпечення особливе місце займають програмні системи, які використовують символьні (алгебраїчні) перетворення та методи комп'ютерної алгебри. Даними таких програм є математичні формули, а алгоритми (алгоритми комп'ютерної алгебри) мають розв'язувати задачі шляхом перетворень формул, тобто символьними перетвореннями. Звичайно, розв'язками задач є також формули.

Типовим елементарним прикладом такої задачі є задача розв'язання квадратного рівняння в радикалах – задача шкільної алгебри. Спрощення відомої зі шкільного курсу алгебри формули коренів квадратного рівняння, яку потрібно виконати для отримання відповіді, робить алгоритм нетривіальним. Зауважимо, що наближені обчислення, реалізовані звичайно в алгоритмах операцій з дійсними числами у мовах програмування, для задач такого роду є непридатними. Обчислення мають бути точними. Точні обчислення потребують реалізації так званих конструктивних числових типів даних, операції яких мають бути реалізовані точними алгоритмами.

Математичні теореми часто носять абстрактний характер. Це, наприклад, так звані теореми існування. Так, теорема Коші про корені довільного полінома стверджує, що поліном степеня n з дійсними (або комплексними) коефіцієнтами має n комплексних коренів, якщо враховувати і їх кратність. Це – типова теорема існування, оскільки про алгоритм обчислення цих коренів мова не йде і доведення цієї теореми алгоритму обчислення коренів не містить. Отже, може бути поставлена задача пошуку комплексних коренів полінома, заданого своїми коефіцієнтами. Це – типова задача комп'ютерної алгебри. У цьому посібнику розглядаються методи програмування символьних обчислень, основні алгоритми комп'ютерної алгебри та приклади застосування цих методів у прикладних задачах.

Глава 1 посібника містить попередні відомості про предмет вивчення, а саме: основні відомості про комп'ютерну арифметику (елементарну теорію чисел), аксіоматичні визначення та приклади конструктивних класичних

алгебр, які будуть використовуватися у подальшому. Автор підкреслює методологічну єдність аксіоматичного та конструктивного підходів до формулювання абстрактних алгебраїчних теорій та визначення алгебраїчних структур з конструктивно визначеними носіями та сигнатурами операцій, які по суті є моделями цих теорій.

Главу 2 присвячено основам технологій алгебраїчного програмування, побудованих на теорії систем переписувань. Правило переписування визначає умовне або безумовне перетворення виразу, що обробляється. Основною структурною одиницею системи алгебраїчного програмування є система правил переписування (rewriting rules system). Застосування системи переписувань до алгебраїчного виразу є кроком алгебраїчного алгоритму, що полягає у перетворенні цього виразу за одним з правил даної системи.

Навчальний матеріал цієї глави містить, по-перше, основні елементарні відомості про ідеологію переписувань, що є фундаментом алгоритмічного підходу до розв'язання задач комп'ютерної алгебри, і, по-друге, вступ до системи алгебраїчного програмування APS (**A**lgebraic **P**rogramming **S**ystem) та мови програмування високого рівня APLAN системи програмування APS.

У **Главі 3** викладено математичні моделі та методи програмування алгебраїчних обчислень**.** Як відомо, математичною моделлю даних мови програмування є багатосортна алгебра або багатосортна алгебраїчна система. Окремі типи даних називають сортами цієї системи. Аргументами та результатами її операцій та відношень є дані різних сортів. Загальний методологічний підхід посібника, запропонований для реалізації такої системи, заснований на представленні її математичної моделі у вигляді ієрархій окремих її сортів, побудованих у чітко визначених термінах відношень між сортами. Це відношення наслідування, розширення, ізоморфізму або гомоморфізму сортів. Окрему роль у побудові багатосортної алгебри як ієрархії сортів відіграє метод параметризації та напівконструктивні означення алгебр. Відзначимо, що наслідування як відношення між сортами давно і широко використовується в математиці. Наприклад, у визначенні

«*Групою називається напівгрупа, кожний елемент якої має обернений елемент*» поняття Група визначено як результат наслідування поняття Напівгрупа. Важливу, вирішальну роль у побудові ієрархії сортів грає поняття конструктивного розширення алгебр. По суті це метод побудови сорту через перевантаження операції базового сорту. Нарешті, ізоморфізми або гомоморфізми є математичними моделями перетворення та приведення типів. Часто конструктивне по суті визначення деякої алгебри (сорту) використовує абстрактну, тобто аксіоматично визначену алгебру у якості параметра. Відомим прикладом такої конструкції є визначення арифметичного n-вимірного векторного простору W над (деяким) полем F. Елементами W є вектори виду $(a_1,a_2,...,a_n)$, де $a_i \in F$. Операції над векторами виконуються покоординатно:

$$(a_1, a_2,..., a_n) + (b_1, b_2,..., b_n) = (a_1 + b1,\ a_2 + b_{2,}..., a_n + b_n).$$

Отже алгоритм визначення векторних операцій зводиться до алгоритмів операції поля F.

Результатом побудови ієрархії сортів має бути повний перелік інтерпретаторів операції даної багатосортної алгебри. Якщо багатосортна алгебра вже побудована і системи аксіом кожного з сортів відомі, алгоритми виконання операцій можна використати для перевірки виконуваності аксіом кожного сорту. Цей підхід по суті визначає метод верифікації окремих сортів та усієї багатосортної алгебри у цілому.

Главу 4 присвячено методам комп'ютерної алгебри в задачах шкільної тригонометрії. Методи та алгоритми цієї глави є комплексним прикладом застосування підходу попередньої глави до реалізації тригонометричних обчислень. Показано, що алгоритми тригонометричних обчислень потребують реалізації чотирьох відносно незалежних ієрархій сортів багатосортних алгебр. Це алгебра раціональних тригонометричних виразів, алгебра коефіцієнтів раціональних тригонометричних виразів, алгебра аргументів тригонометричних функцій та алгебра розв'язків тригонометричних рівнянь

та нерівностей. Кожна з цих багатосортних алгебр є достатньо складною. Зокрема, алгебра коефіцієнтів раціональних тригонометричних виразів має містити обчислення в радикальних та кругових розширеннях поля раціональних чисел.

Глава 5 містить викладення основних, найбільш затребуваних алгоритмів комп'ютерної алгебри. Це ефективні алгоритми множення цілих чисел та поліномів, алгоритми факторизації поліномів однієї змінної, алгоритми обчислення базисів Гребнера та їх застосування в задачах комп'ютерної алгебри, зокрема, в задачах теорії Галуа, а також алгоритм Штурма відділення та уточнення дійсних коренів полінома з дійсними коефіцієнтами. Як відомо, основною проблемою алгебри є проблема розв'язання системи алгебраїчних рівнянь. Перелічені вище алгоритми направлені на алгоритмічне розв'язання окремих задач цієї комплексної проблеми.

Глава завершується викладенням алгоритму розв'язання системи лінійних нерівностей з n невідомими. Як відомо, розв'язком системи лінійних нерівностей є така опукла область арифметичного n-вимірного лінійного простору, кожна точка якої є частинним розв'язком системи. Отже, розв'язком системи лінійних нерівностей має бути формула, що однозначно визначає її область розв'язків. У стандартному підході розв'язок представляється у вигляді множини вершин (0-вимірних граней) цієї області. Підхід, який застосовується у цій главі, полягає у наступному: розв'язок системи лінійних нерівностей представлено у вигляді розбиття області розв'язків у суму так званих трапецоїдів. Оскільки таке представлення є канонічною формою системи лінійних нерівностей, його можна вважати розв'язком системи. Параграф завершується ефективним алгоритмом обчислення множини 0-вимірних вершин області розв'язків системи.

Главу 6 присвячено одній із задач статичного аналізу програм – задачі генерації поліноміальних інваріантів комп'ютерних програм. Алгоритми цієї

глави повною мірою використовують алгебраїчні обчислення та методи комп’ютерної алгебри.

Глава 1. Вступ до комп'ютерної алгебри

1.1. Основи комп'ютерної арифметики

1.1.1. Основні операції комп'ютерної арифметики.

Система команд будь-якого комп'ютера містить команди виконання арифметичних операцій над числами.

Комп'ютерною арифметикою називають розділ комп'ютерної науки, який вивчає особливості виконання арифметичних обчислень у комп'ютерах.

Числа у комп'ютері представлені у двійковій системі числення. Ця система дуже схожа на десяткову систему, якою ми звикли користуватися. І одна, і інша є позиційними системами, оскільки вони використовують один і той же принцип представлення чисел.

Позиційні системи. Натуральне число у позиційний системі числення представлено послідовністю цифр. Кожна цифра позначає одне з чисел $0,1,...,p-1$. Число p називають основою системи числення. Десяткова система числення використовує цифри $0,1,...,9$. Для неї $p=10$. Двійкова система числення використовує цифри $0,1$, а $p=2$. «Вага» цифри, тобто її внесок у величину числа, залежить від позиції (розряду) цифри.

Якщо число N представлено (записано) у десятковій системі послідовністю цифр $a_k\, a_{k-1} \ldots a_1\, a_0$ ($N_{10} = a_k\, a_{k-1} \ldots a_1\, a_0$), то $N = a_k 10^k + a_{k-1} 10^{k-1} + \ldots + a_1 10 + a_0$.

Якщо число N представлено у системі числення за основою p послідовністю цифр $a_k\, a_{k-1} \ldots a_1\, a_0$ то $N = a_k p^k + a_{k-1} p^{k-1} + \ldots + a_1 p + a_0$.

Розглянемо особливості представлення цілих чисел. Арифметичні команди комп'ютера оперують з двійковими числами, кожне з яких розміщено у комірці пам'яті розміром у один або декілька байтів. У комп'ютерах з 32-бітним процесором ціле число, зокрема, займає 32 біта (або 4 байти). При цьому старший біт є бітом знаку числа, а інші біти містять цифри числа (див. рис. 1.1). Значення $s=0$ означає знак плюс (+, а $s=1$ – знак мінус (-).

s	**A_{31}**	**a_{30}**	**.**	**.**	**.**	**a_1**	**a_0**

Рис 1.1. Представлення цілого числа в пам'яті комп'ютера

Таким чином, найбільше ціле число, яке може бути представлено 32 бітами, дорівнює 2^{31}-1. Найменшим є число –(2^{31}-1). (У мові Паскаль так представлені цілі числа типу *LongInt*). Аналогічно, цілі числа представляються у двох байтах пам'яті (тип *Integer*) і у одному байті пам'яті (тип *Byte*).

Арифметичні операції над цілими числами виконуються процесором за правилами арифметики багаторозрядних чисел (додавання у стовпчик, множення у стовпчик, ділення з остачею кутком і т.і.). Звичайно, що введення - виведення чисел користувачем має здійснюватися у десятковій системі. Це означає, що основними алгоритмами дій з цілими числами є алгоритми

перекладу чисел з однієї позиційної системи до іншої, алгоритми виконання операцій додавання, віднімання, множення, возведення у степінь та обчислення неповної частки та остачі. Логічними операціями є операції порівняння цілих чисел на «менше», «більше» та «дорівнює». Значеннями цих операцій є логічні значення *True, False*.

Арифметика необмеженої розрядності. Більшості звичайних комп'ютерних програм для правильного виконання алгоритмів вистачає діапазону цілих чисел типів *Integer, LongInt.* Однак, при виконанні арифметичної операції її результат може вийти за діапазон типу. Це призведе до помилки у процесі виконання програми. До того ж, практична інформатика користується такими алгоритмами, в яких треба виконувати арифметичні дії з цілими числами, кількість цифр яких сягає тисяч, десятків тисяч. Це, наприклад, алгоритми криптографії (RSA-алгоритми [1-3]). Ця особливість стосується і алгоритмів комп'ютерної алгебри. Тому важливою задачею комп'ютерної арифметики є задача реалізації арифметичних обчислень з цілими числами необмеженої розрядності (арифметика довгих цілих).

Арифметика довгих цілих також користується представленням чисел у позиційній системі числення, але цифрами цієї системи, як правило, є числа типу *LongInt*. Таким чином, в арифметиці довгих цілих сучасних спеціалізованих бібліотек $p = 2^{31} - 1$.

Для того, щоб представити ціле число, необхідно представити його знак та абсолютну частину, яку називають мантисою. Для ефективної реалізації логічних операцій дуже корисним виявляється також знання кількості цифр числа. Мантису числа представляють у вигляді динамічного масиву (тип *DVector*). Цифри числа у масиві розташовують у порядку зростання розрядів. Кількість цифр та знак числа представляють цілим числом Length. Знак Length є знаком представленого числа, а абсолютна величина Length – кількість цифр числа.

```
Type
LongNum = record
  Length: LongInt;   {кількість цифр та знак числа}
  Mantissa: DVector; {мантиса числа}
End;
```

Часто у прикладних програмах арифметику довгих цілих можна замінити на арифметику цілих підвищеної розрядності. У цьому випадку мантису числа представляють у статичному масиві (тип *Vector*).

```
Const MaxLength = 100; BaseP = 2147483648;
Type
  Vector = array[0..MaxLength] of LongInt;
  LongNum = record
 Length: Integer;    {кількість цифр та знак числа}
      Mantissa: Vector;   {мантиса числа}
End;
```

Наведемо процедуру додавання двох натуральних чисел A та B:

```
Procedure Add(var A, B, R: LongNum)
Var i: Integer; C, W: LongInt;
Begin
  R.Length := Max(A.Length, B.Length); W := 0;
  For i := 0 to  R.Length  do begin
     C := A.Mantissa[i] + B.Mantissa[i] + W;
     R.Mantissa[i] := C mod BaseP;
    W := C div BaseP
  End;
  If W = 1 Then begin
     R.Length := R.Length + 1;
     R.Mantissa[R.Length]:=1
  End
End;
```

Наведемо функцію порівняння цілих чисел A та B на «менше».

```
Function Less (var A, B: LongNum): Boolean;
Var i: Integer;
Begin
  If A.Length = B.Length Then begin
    i:= A.Length;
    While(A.Mantissa[i]=B.Mantissa[i])and(i>=0) do i:=i-1;
    If i < 0 Then Less := False {числа рівні}
      Else
       Less:=(A.Mantissa[i]<B.Mantissa[i])and(A.Length>0)
       or (A.Mantissa[i] > B.Mantissa[i]) and (A.Length > 0)
  end
  Else Less := (A.Length < B.Length)
End;
```

Розглянуті приклади та пояснення до них наочно демонструють техніку програмування бібліотеки довгих цілих. Деякі бібліотеки довгих цілих чисел можна отримати на умовах вільного або умовно вільного розповсюдження (наприклад, бібліотека **GMP**).

1.1.2. Арифметика раціональних чисел

Комп'ютерна алгебра – розділ комп'ютерної науки, що вивчає алгоритми розв'язання математичних задач, розв'язком яких є формула.

В задачах комп'ютерної алгебри, на відміну від задач наближених обчислень, використовують точні алгоритми, засновані на перетвореннях формул (символьних перетвореннях). Математики, фізики, інженери та спеціалісти інших галузей у своїй діяльності широко використовують спеціальні математичні системи. Найбільш відомими серед професіоналів є математичні системи *Mathematica, Maple, Mathcad, Mathlab*. Математичні системи використовують також у навчальному процесі. Найбільш відомою

математичною системою серед вчителів математики та учнів є система *Derive*. Першими вітчизняними математичними системами навчального призначення є ПМК “Системи лінійних рівнянь”, “ТерМ”, “Web Almir”.

Математичні системи мають оперувати не лише цілими числами, а і дробами, радикалами, комплексними числами тощо. Реалізацію обчислень з раціональними числами, представленими у вигляді звичайних дробів, називають арифметикою раціональних чисел.

Алгоритми раціональних обчислень засновані на “шкільних” правилах виконання арифметичних операцій. Наприклад, правило складання дробів спирається на формулу $\frac{a}{b}+\frac{c}{d}=\frac{a\cdot d+b\cdot c}{b\cdot d}$. Є лише одне перетворення, яке треба додатково виконати: треба скоротити чисельник та знаменник результату на їх найбільший спільний дільник. Позначимо цю операцію скорочення через *reduce*. Аргументами *reduce* є два натуральних числа (чисельник і знаменник дробу, який треба скоротити), а результатом – відповідний нескорочуваний дріб. Тоді правило додавання дробів можна уточнити:

$$\frac{a}{b}+\frac{c}{d}=reduce(a\cdot d+b\cdot c,b\cdot d).$$

Залишилося визначити правила додавання цілих чисел та дробів, і ми отримаємо формули, за якими можна програмувати операцію додавання дробів:

$$\frac{a}{b}+\frac{c}{d}=reduce(a\cdot d+b\cdot c,b\cdot d),\ a+\frac{c}{d}=\frac{a\cdot d+c}{d},\ \frac{a}{b}+c=\frac{a+b\cdot c}{b}.$$

Аналогічно визначають інші арифметичні та логічні операції над дробами. Розглянемо більш детально операцію *reduce*. Вона є ключовою, оскільки скорочувати дроби треба практично у кожній операції. Ефективність алгоритмів дій з раціональними числами визначальною мірою залежить від ефективності алгоритму обчислення найбільшого спільного дільника двох натуральних чисел (НСД).

Алгоритм Евкліда. Алгоритм обчислення НСД двох чисел був відомий ще геніальному грецькому філософу та математику Евкліду. Його можна описати таким чином:

1.Розглянь два числа A и B.
2.Знайди остачу R від ділення більшого числа на менше.
3.Якщо R = 0, то менше число є результатом, тобто НСД(A, B)
інакше R підстав на місце більшого числа та повтори обчислення

```
    Function GCD(A, B: LongNum):LongNum; {GCD-Greatest Common
Divisor}
    begin {A > 0, B > 0}
     While (A<>0)and(B<>0) do
        if A > B
```

```
      then A := A mod B
      else B := B mod A;
 GCD := A+B  {A або B дорівнює нулю}
 end;
```

Існує багато варіантів алгоритму Евкліда. Найпростіший з них замість обчислень остач від ділення одного числа на інше використовує віднімання.

```
Function GCD(A, B: LongNum): LongNum;
begin           {A > 0, B > 0}
  While (A<>B) do
   if A > B
 then A := A - B
 else B := B - A;
 GCD := A  {A дорівнює B}
end;
```

Лаконічним є рекурсивний варіант алгоритму, заснований на співвідношенні НСД(A, B) = НСД(B mod A, A mod B)

```
Function GCD (A, B: LongNum): LongNum;
begin {A > 0, B > 0}
  If  B <> 0
    then GCD := GCD(B mod A, A mod B)
    else GCD := A
End;
```

Зауважимо, що операції *mod* та *div* виконуються процесором досить повільно. Тому важливо побудувати алгоритм обчислення НСД, вільний від цього недоліку. Таким алгоритмом є бінарний алгоритм Евкліда. Ідея цього алгоритму полягає у наступному:

- Якщо обидва числа є парними, їх можна поділити на 2, помноживши результат на 2:

  $(a \bmod 2 = 0) \,\&\, (b \bmod 2 = 0) \rightarrow GCD(a,b) = 2GCD(a \; div \; 2,\; b \; div \; 2)$

- Якщо лише одне з чисел є парним, його можна скоротити на 2:

  $(a \bmod 2 = 0) \,\&\, (b \bmod 2 = 1) \rightarrow GCD(a,b) = GCD(a \; div \; 2,\; b)$

  або

  $(a \bmod 2 = 1) \,\&\, (b \bmod 2 = 0) \rightarrow GCD(a,b) = GCD(a,\; b \; div \; 2)$

- Якщо обидва числа є непарними, від більшого з чисел треба відняти менше. Тоді їх різниця стане парною:

  $(a \bmod 2 = 1) \,\&\, (b \bmod 2 = 1) \rightarrow ((a > b) \rightarrow GCD(a,b) = GCD((a-b) \; div \; 2, b)$
  $else \; GCD(a,b) = GCD((b-a) \; div \; 2, a)$

- Останні дві дії треба повторювати, доки обидва числа не дорівнюють нулю.

```
Function BinGCD( A, B: LongNum): LongNum;
Var D: LongNum;
begin {A > 0, B > 0}
D := 1;
While (A mod 2 = 0) and  (B mod 2 = 0)  do begin
   A := A div 2; B  :=  B div 2; D := 2*D
end;  {Визначено найбільший спільний дільник D виду 2^k }
While (A <>0) and (B <> 0) do begin
While A mod 2 = 0 do A := A div 2;
While B mod 2 = 0 do B := B div 2;
 If A > B then A := A - B  else B := B - A
end;
 BinGCD := (A+B)*D
end;
```

Таким чином, бінарний алгоритм Евкліда використовує лише операції обчислення остачі та неповної частки від ділення на 2. Але, $a \bmod 2$ – це остання цифра a, а $a \; div \; 2$ отримують операцією зсуву числа a на 1 розряд вправо. Обидві дії виконуються за 1 такт процесора.

Розширений алгоритм Евкліда разом з НСД натуральних чисел a,b знаходить такі цілі числа u,v, що $d = uA + vB$.

```
Procedure ExtendedGCD(A, B: LongNum; var d,u,v: LongNum);
Var
  x, y: LongNum;
  u1, u2, v1, v2: LongNum;
begin           {A > 0, B > 0}
 x := A; y := B;
 u1:= 1; v1 := 0; {A = u1*A + v1*B}
 u2:= 0; v2 := 1; {B = u2*A + v2*B}
While (x <> y) do
   if x > y
     then begin
       x := x - y;
       u1 := u1 - u2;
       v1 := v1 - v2; {x = u1*A + v1*B}
     end
     else begin
       y := y - x;
       u2 := u2 - u1;
       v2 := v2 - v1;  {y = u2*A + v2*B}
     end
 d := x  {x = y, d = x}
 u := u1; v := v1
end;
```

1.1.3. Прості числа

Натуральне число p називається простим, якщо воно ділиться тільки на 1 и на себе. Алгоритми розв’язання різних задач на прості числа знаходяться

у центрі уваги спеціалістів з комп'ютерної алгебри. Ми вже згадували про криптографію. Центральними задачами криптографії є задача розкладу числа на множники (факторизація числа) і задача генерації великих простих чисел. Сучасні ефективні алгоритми розв'язання цих задач спираються на досягнення теорії чисел та комп'ютерної алгебри. Цікавим, однак, є те, що перші кроки у встановленні цих алгоритмів були зроблені видатними вченими минулого. Зараз ми розглянемо алгоритм пошуку простих чисел, відомий як *решітка Ератосфена***.**

Алгоритм Ератосфена будує множину усіх простих чисел інтервалу [2...n]. Він полягає у наступному:

1. Будуємо множину A усіх натуральних чисел від 2 до n. Множина простих чисел P є пустою.

2. Нехай q – найменше число множини A. "Просіваємо" множину A крізь решето з кроком q, тобто

- додаємо q до множини P простих чисел;
- видаляємо з A усі числа, кратні q, починаючи з q;

Процес просівання повторюється, доки множина A не є пустою.

```
Type NumSet = Set of LongNum;
Procedure EratosphenGrid (N: LongNum);
Var
A, P: NumSet;
 q, r: LongNum;
Begin
   Init(A, N);
   P := EmptySet; { пуста множина }
   Repeat
       q := MinElement(A);
       P := P + {q};
       For r := q  to  N div q  do  A := A - {r*q};
   Until A = EmptySet
End;
```

Звичайно, задачі та алгоритми комп'ютерної арифметики, які ми розглянули, є лише вступом до широкого кола важливих та цікавих проблем комп'ютерної алгебри, які зараз знаходяться у центрі уваги вчених та програмістів.

1.1.4. Китайська теорема про остачі

Нагадаємо, що арифметичною прогресією називається послідовність чисел $\{a_n\}$, визначена рекурсивно формулами

$$a_0 = b,\ a_{i+1} = a_i + d,\ \ i = 0,1,...$$

Число b називають початковим членом прогресії, а число d - різницею прогресії. i -тий член прогресії обчислюється за формулою $a_i = b + id$.

Нескінченну арифметичну прогресію виду $a_i = b + id$ будемо позначати через $[b,d]$, а її скінченний початок - $a_0 = b,\ a_2,...,a_{k-1} = b + (k-1)d$ - через $[b,d]^{(k)}$.

Поняття арифметичної прогресії можна узагальнити, якщо допустити декілька початкових членів. А саме: нехай $B = \{b_1, b_2,..., b_m\}$ – скінченна множина. Узагальненою арифметичною прогресією з початковою множиною B і різницею d назвемо послідовність (множину) чисел, визначену формулами

$$B_0 = B,\ A_0 = B_0,\ B_{i+1} = \{b_1 + id, b_2 + id,..., b_m + id\},\ A_{i+1} = A_i \cup B_{i+1},\ i = 1,2,...$$

Узагальнену арифметичну прогресію будемо позначати через $[B,d]$. Теоретико-множинні операції над узагальненими арифметичними прогресіями визначаються формулами

$$[B_1,d] \cup [B_2,d] = [B_1 \cup B_2, d] \quad (1)$$

$$[B_1,d] \cap [B_2,d] = [B_1 \cap B_2, d] \quad (2)$$

Арифметичну прогресію $[b,d]$ можна представити як узагальнену прогресію. Нехай k – довільне натуральне число. Тоді

$$[b,d] = [[b,d]^{(k)}, kd] \quad (3)$$

У подальшому будемо вважати, що b,d - натуральні числа і $b < d$. Такі прогресії будемо називати натуральними. Тоді для будь-якого натурального n

$$b = a_n \ mod\ d$$

і арифметична прогресія $[b,\ d]$ – множина усіх розв'язків рівняння

$$x\ mod\ d = b. \quad (4)$$

Рівність виду (4) називається порівнянням або конгруенцією за модулем d. Для позначення конгруенцій використовують наступну формулу:

$$x \equiv b (\mathrm{mod}\ b)$$

Теорема 1.1. Якщо $[b_1,d_1], [b_2,d_2]$ – натуральні прогресії і d_1, d_2 – взаємно прості, існує таке $b < d_1 \cdot d_2$, що

$$[b_1,d_1] \cap [b_2,d_2] = [b,\ d_1 \cdot d_2].$$

Доведення. Застосовуючи (3), (2) отримаємо:

$$[b_1,d_1]=[[b_1,d_1]^{(d_2)},d_1d_2],\ [b_2,d_2]=[[b_2,d_2]^{d_1},d_1d_2]$$

$$[b_1,d_1]^{d_2}\cap[b_2,d_2]^{d_1}=[b_1,d_1]=[[b_1,d_1]^{(d_2)}\cap[b_2,d_2]^{d_1},d_1d_2]$$

Таким чином, якщо $[b_1,d_1]^{d_2}\cap[b_2,d_2]^{d_1}\neq\varnothing$, мають існувати такі натуральні числа $k_1,k_2,k_1<d_1d_2$, $k_2<d_1d_2$, що $b_1+k_1d_1=b_2+k_2d_2$, або $k_1d_1-k_2d_2=b_2-b_1$. Однак, за розширеним алгоритмом Евкліда, існують такі натуральні числа u,v, що $ud_1-vd_2=1$. Для визначеності будемо вважати, що $b_2\geq b_1$. Домноживши обидві частини цієї рівності на b_2-b_1, отримаємо:
$(b_2-b_1)ud_1-(b_2-b_1)vd_2=b_2-b_1$. Отже, $K_1=(b_2-b_1)u$, $K_2=(b_2-b_1)v$, $k_1=K_1 \bmod d_1d_2$, $k_2=K_2 \bmod d_1d_2$.

Наслідок 1 (інше формулювання теореми *1). Якщо d_1,d_2 взаємно прості, система *лінійних модулярних рівнянь*

$$\begin{cases} x\equiv b_1(mod\ d_1), \\ x\equiv b_2(mod\ d_2). \end{cases} \tag{5}$$

має єдиний розв'язок за модулем $d=d_1\cdot d_2$.

Наслідок 2 (узагальнення наслідку 1). Якщо $d_1,d_2,...d_n$ попарно взаємно прості, система лінійних модулярних рівнянь

$$\begin{cases} x\equiv b_1(mod\ d_1)_1, \\ x\equiv b_2(mod\ d_2)_2. \\ ..... \\ x\equiv b_n(mod\ d_n). \end{cases} \tag{6}$$

має єдиний розв'язок за модулем . $d=d_1\cdot d_2\cdot ...\cdot d_n$

Результати теореми 1.1 та наслідків можна використовувати у так званих модулярних обчисленнях.

Наслідок 3. Нехай $d=d_1\cdot d_2\cdot ...\cdot d_n$, де $d_1,d_2,\ ...,d_n$ – попарно взаємно прості числа і a – довільне натуральне число з інтервалу $[0;p-1]$. Обчислимо

$$a_1=a \bmod d_1,\ a_2=a \bmod d_2,\ ...,\ a_n=a \bmod d_n$$

Тоді відповідність $\varphi_d: a\to(a_1,a_2,...,a_n)$ є взаємно-однозначною. Інакше кажучи, число a з інтервалу $[1,d]$, $d=d_1\cdot ...\cdot d_n$ можна відновити за його остачами від ділення на взаємно прості числа $d_1,\ ...,\ d_n$. Неважко показати, що

$$\varphi_d(a+b)=((a_1+b_1) \bmod d_1,\ (a_2+b_2) \bmod d_2,...,(a_n+b_n) \bmod d_n)$$

$$\varphi_d(a\cdot b)=((a_1\cdot b_1) \bmod d_1,\ (a_2\cdot b_2) \bmod d_2,...,(a_n\cdot b_n) \bmod d_n)$$

Теорема 1.1 та її наслідки – це різні формулювання так званої китайської теореми про остачі. На завершення підпункту наведемо простий алгоритм теореми 1.1. Зауважимо, що

$$[b_1, d_1]^{(d_2)} = \{b_1, b_1 + d_1, ..., b_1 + d_2 d_1\} \quad \{b_1 < b_1 + d_1 < ... < b_1 + d_2 d_1\}$$
$$[b_2, d_2]^{(d_1)} = \{b_2, b_2 + d_2, ..., b_1 + d_1 d_2\}, \text{ причому } \{b_2 < b_2 + d_2 < ... < b_1 + d_1 d_2\}$$

Таким чином, задача полягає в обчисленні перетину двох зростаючих послідовностей, якщо відомо, що цей перетин містить рівно один елемент. Наведемо алгоритм обчислення елементів b, d з формули (4).

```
Procedure IntersectSeq(b1,d1,b2,d2: LongNum; var b,d: LongNum);
Var x, y: LongNum;
begin
 x := b1;
 y := b2;
 While x <> y do
   If x<y
     then x := x + d1
     else y := y + d2;
 b := x;
 d := d1*d2
end;
```

Цей алгоритм використовується як допоміжний в алгоритмі обчислення розв'язків системи (6). Нехай в послідовності

$$[b_1, d_1], [b_2, d_2], ..., [b_n, d_n] \quad d_1 < d_2 < ... < d_n. \qquad (7)$$

Обчислимо $b, d = d_1, d_2$ з перетину $[b_1, d_1] \cap [b_2, d_2] = [b, d_1 \cdot d_2]$. Виключимо $[b_1, d_1]$, $[b_2, d_2]$ з послідовності (8) обчислимо $[b, d_1 \cdot d_2]$ та вставимо у отриману послідовність $[b, d_1 \cdot d_2]$ з збереженням порядку (7). Отримаємо

$$[b_3, d_3], [b_4, d_4], ..., [b, d_1 d_2], ..., [b_n, d_n], \quad d_3 < d_4 < ... < d_1 d_2 < ... < d_n$$

Цю операцію треба повторити $n-1$ разів, отримавши в результаті $[b, d_1 d_2 ... d_n]$. Число b є результатом алгоритму. Зауважимо, що реалізацією цього алгоритму фактично є черга з пріоритетами.

1.2. Аксіоми і приклади класичних алгебр

У цьому параграфі наведено основні, початкові відомості про алгебри, розглянуті в книзі, з метою зробити викладення матеріалу максимально незалежним. Викладення матеріалу не претендує на повноту. Усі алгебраїчні поняття, що розглядаються у книзі, вивчаються в університетських курсах загальної (вищої) алгебри. Книги, які ми рекомендуємо в посиланнях у тексті, допоможуть читачеві, який бажає усунути цей недолік.

1.2.1. Алгебри, аксіоматичні та конструктивні визначення

Основним поняттям алгебри є поняття алгебраїчної операції. Алгебраїчна операція на множині A – це функція одного або декількох аргументів

$y = f(x_1,...,x_n)$, аргументи і значення якої належать множині A, яка називається носієм операції . Число n називається арністю операції. Операції арності 1 (з одним аргументом) називають унарними, а операції арності 2 – бінарними.

Алгеброю називається множина A з визначеними на ній однією або декількома операціями. Множина A називається носієм алгебри.

Алгебра вивчає властивості алгебраїчних операцій. Властивості операцій можна визначити на абстрактному рівні – за допомогою системи аксіом, а потім виводити різні наслідки з цієї системи аксіом. Абстрактні властивості алгебри не залежать від її носія. Проте, будь-який конкретний екземпляр алгебри, що використовується в математичній практиці, вимагає точного конструктивного визначення її носія та, як правило, алгоритмів виконання операцій, тобто алгоритмів обчислення значення функції $y = f(x_1,...,x_n)$. Алгебри, визначені таким чином, називають конструктивними. Для конструктивної алгебри аксіоми відіграють роль властивостей, які потребують доведення.

Алгебраїчні операції можуть бути визначені як повністю, так і частково. Операція додавання на множині натуральних чисел, наприклад, визначена повністю, а операція віднімання – частково. Нарешті, в багатьох практично важливих випадках алгебраїчні операції можуть бути визначені для різнотипних аргументів і значень. Наприклад, в курсі алгебри операція піднесення до степеня $b = a^n$ визначається спочатку тільки для натуральних n.

Нехай A – носій операції $y = f(x_1,...,x_n)$ і $B \subset A$. Якщо з того, що $a_1,...,a_n \in B$ слідує, що $f(a_1,...,a_n) \in B$, то кажуть, що операція $y = f(x_1,...,x_n)$ замкнута на множині B. Якщо всі операції алгебри А з носієм A замкнуті на множині B, кажуть, що алгебра В з носієм B є підалгеброю алгебри В. У цьому випадку А називається розширенням В.

Наприклад, операція додавання на множині Z цілих чисел замкнута на множині N натуральних чисел. Тому алгебра $\mathrm{N} =< N, + >$ – підалгебра алгебри $\mathrm{Z} =< Z, + >$.

Загальним питанням алгебри присвячені книги [4-8].

1.2.2. Напівгрупи

Напівгрупою називається алгебра S з однією бінарною операцією $c = a \cdot b$ що задовольняє аксіому асоціативності:

$$(a \cdot b) \cdot c = a \cdot (b \cdot c).$$

Типом операції $y = f(x_1,...,x_n)$ ми будемо називати формулу виду $T_1 \times T_2 \times ... \times T_n \to T$, де T_j позначає алгебру аргументу $x_j (x_j \in T_j)$, а T – алгебру результату $y (y \in T)$. У цих позначеннях напівгрупова операція має тип $S \times S \to S$.

Приклади напівгруп :

1. Напівгрупа слів у алфавіті $\Sigma =< a,b,... >$. Алфавітом називають скінченну упорядковану множину символів. Напівгрупу слів в алфавіті Σ позначимо через Σ^*. Елементами Σ^* є слова – ланцюжки символів алфавіту Σ. Множення двох слів $p = u_1u_2...u_k$, $q = v_1v_2...v_m$ здійснюється дописуванням слова q праворуч до слова p :

$$p \cdot q = u_1u_2...u_kv_1v_2...v_m \ .$$

2. Напівгрупа M^A відображень на множині A. Нехай, наприклад, $A = \{a,b,c\}$. Тоді кожне відображення можна записати у вигляді таблиці, що складається з двох рядків. Перший рядок містить елементи A, а другий – відображення цих елементів. Наприклад:

$\alpha = \begin{pmatrix} a & b & c \\ c & a & a \end{pmatrix}$ визначає відображення $\alpha : A \to A$ рівностями $\alpha(a) = c, \alpha(b) = a, \alpha(c) = a$.

Операція множення $\alpha \cdot \beta$ елементів α, β напівгрупи M^A визначається співвідношенням

$$\forall x \in A \ (\alpha \cdot \beta)(x) = \beta(\alpha(x)) \, .$$

Легко підрахувати, що ця напівгрупа містить 27 елементів.

3. Множина $N = \{1,2,3,...\}$ натуральних чисел відносно арифметичної операції множення.

Одиниця

Одиницею (нейтральним елементом відносно напівгрупової операції) називається елемент , що задовольняє аксіому

$$a \cdot e = e \cdot a = a$$

Нуль

Нулем (анулюючим елементом відносно напівгрупової операції) називається елемент, що задовольняє аксіому

$$a \cdot O = O \cdot a = O$$

Властивість комутативності

Напівгрупа називається комутативною, якщо виконується аксіома

$$a \cdot b = b \cdot a$$

Приклад комутативної напівгрупи

Напівгрупа $[X]$ мономів над алфавітом $X=\{x_1,...,x_n\}$. Елементи $[X]$ задані визначенням $[X]=\{x_1^{k_1}x_2^{k_2}...x_n^{k_n}, k_1,...,k_n \in N\}$. Операція множення задана визначенням

$$x_1^{k_1}x_2^{k_2}...x_n^{k_n}\cdot x_1^{m_1}x_2^{m_2}...x_n^{m_n}=df=x_1^{k_1+m_1}x_2^{k_2+m_2}...x_n^{k_n+m_n}.$$

Символи $x_1,...,x_n$ називаються твірними елементами напівгрупи $[X]$.

Напівгрупи вивчає теорія напівгруп [9-10] .

У різних розділах алгебри напівгрупова операція позначається різними символами.

1. В арифметиці цілих чисел напівгруповими операціями є операції додавання і множення.

2. В математичній логіці операції кон'юнкції і диз'юнкції також асоціативні, і, отже, множини термів алгебри логіки утворюють комутативну напівгрупу і відносно кон'юнкції, і відносно диз'юнкції. Операції кон'юнкції і диз'юнкції задовольняють аксіомам ідемпотентності :

$$a\,\&\,a=a,\ a\vee a=a.$$

1.2.3. Групи

Групою G називається напівгрупа з одиницею, в якій для кожного елемента існує йому зворотний. Визначальна аксіома зворотного елемента:

$$a\cdot a^{-1}=a^{-1}\cdot a=e$$

Приклади груп:

1. Група слів у «подвійному» алфавіті $\Sigma\Sigma=<a,a',b,b,',...>$. Подвійний алфавіт $\Sigma\Sigma$ разом з кожним символом $a\in\Sigma\Sigma$ містить його символ-двійник $a'\in\Sigma\Sigma$. При цьому для кожного $a\in\Sigma\Sigma$, за означенням, виконується співвідношення $a\cdot a'=e$, де $e\notin\Sigma\Sigma$ – одиниця групи. Легко бачити, що $a^{-1}=a', (a')^{-1}=a.$ Для слова $p=a_1a_2...a_{k-1}a_k$ зворотним є слово $p^{-1}=a_k^{-1}a_{k-1}^{-1}...a_1^{-1}$.

2. Група взаємно-однозначних відображень на множині A. Як і для напівгрупи відображень на множині A, взаємно-однозначне відображення задається таблицею. Наприклад: $\alpha=\begin{pmatrix} a & b & c \\ c & a & b \end{pmatrix}$. Такі таблиці називають підстановками, а групи – групами підстановок. Якщо множина A скінченна, позначення її елементів можна уніфікувати, вважаючи, що множина з n елементів має вигляд $A=\{1,2,...,n\}$. Верхній рядок таблиці можна опустити, записуючи підстановку у вигляді $\alpha=(j_1, j_2,..., j_n)$, де серед чисел $j_1, j_2,..., j_n$ кожне з чисел $1,2,...,n$ зустрічається рівно один раз. Група підстановок з n елементів називається симетричною групою і позначається через S_n. Порядок (кількість елементів) групи S_n дорівнює $n!$.

Унарна операція визначення зворотного елемента може бути використана для визначення бінарної операції ділення:

$$a/b \overset{df}{=} a \cdot b^{-1}.$$

У визначенні групи замість операції зворотного елемента може використовуватися операція ділення. Тому групи називають напівгрупами з діленням. Комутативні групи називають абелевими на честь норвезького математика Н.Х. Абеля. Групи вивчає теорія груп [11 ,12].

1.2.4. Кільця

Кільцем R називається алгебра з двома бінарними операціями – додаванням і множенням. Обидві операції кільця мають тип $R \times R \to R$. Відносно операції додавання кільце є абелевою групою. Цю групу називають адитивною групою кільця. Нейтральний елемент цієї групи називають нулем і позначають символом 0, оскільки 0 є анулюючим елементом відносно операції множення. Відносно операції множення кільце є напівгрупою. Операцію множення зазвичай позначають точкою: $a \cdot b$. Цю напівгрупу називають мультиплікативною напівгрупою кільця. Кільцеві операції задовольняють аксіомам дистрибутивності :

$$a \cdot (b + c) = a \cdot b + a \cdot c, \quad (b + c) \cdot a = b \cdot a + c \cdot a$$

Нуль кільця є анулюючим елементом відносно операції множення:

$$a \cdot 0 = 0 \cdot a = 0.$$

Приклади кілець:

1. Множина Z цілих чисел.

Носій алгебри – множина цілих чисел, записаних, наприклад, у p-ічній системі числення. Арифметичні операції задаються таблицями додавання і множення, а також правилами виконання операцій над багаторозрядними числами, відомими з дитинства.

2. Множина поліномів $Z[x]$ змінної x з цілими коефіцієнтами.

У стандартному вигляді поліноми записуються таким чином: $\alpha_0 x^n + \alpha_1 x^{n-1} + ... + \alpha_n$. Числа $\alpha_0, \alpha_1, ..., \alpha_n$ є коефіцієнтами полінома. Вирази виду $a_i x^{n-i}$ називаються мономами. Поліном є сумою мономів. Операції над поліномами відомі зі шкільного курсу алгебри.

Якщо мультиплікативна напівгрупа кільця містить одиницю, кільце називають кільцем з одиницею. Якщо мультиплікативна група кільця є комутативною, кільце називають комутативним.

3. Кільце N_m остач за модулем m. Нехай m – натуральне число, $m > 1$. На множині $N_m = \{0,1, ..., m-1\}$ визначимо операції кільця "$+_m$","$\cdot_m$" наступними співвідношеннями:

$$a +_m b \overset{df}{=} (a+b) \bmod m, \quad a \cdot_m b \overset{df}{=} (a \cdot b) \bmod m .$$

Можна перевірити, що $N_m < +_m, \cdot_m >$ є комутативнм кільцем з одиницею. Воно називається кільцем остач за модулем m. Якщо m – складене число і $m = a \cdot b$, то $a \cdot_m b = 0$, (див. п. Області цілісності). Якщо m – просте число, кожен ненульовий елемент кільця N_m має зворотний (див. п. 1.2.6 Поля).

Області цілісності. Кільце називається кільцем без дільників нуля (областю цілісності), якщо задовольняється аксіома

$$a \cdot b = 0 \to a = 0 \vee b = 0 .$$

Зазначимо, що ця аксіома не є тотожністю.

Приклад кільця з дільниками нуля:
1. Множина 2×2 матриць над кільцем цілих чисел.
Справді,

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

Приклади областей цілісності :
1. Множина Z цілих чисел.
2. Множина поліномів $Z[x]$ однієї змінної x з цілими коефіцієнтами.

Евклідові кільця
Якщо в кільці E визначено операції
$a \, div \, b$ – операція неповного ділення (ділення з остачею) ,
$a \bmod b$ – операція обчислення остачі,
таке кільце називається евклідовим.
Операції div, mod мають тип $E \times E \to E$.
Можна довести, що евклідові кільця не мають дільників нуля.

Приклади евклідових кілець:
1. Множина Z цілих чисел.
2. Множина многочленів $Z[x]$ однієї змінної з цілими коефіцієнтами.

Кільце поліномів $Z[x, y]$ двох змінних є прикладом області цілісності, але не евклідового кільця .

Кільця вивчає теорія кілець [13-15] .

1.2.5. Булеві алгебри

Булевою алгеброю B називають алгебру з сигнатурою $<\&, \vee, \neg>$ (кон'юнкція, диз'юнкція, заперечення), яка є кон'юнктивною та диіз'юнктивною напівгрупою, що задовольняє аксіомам дистрибутивності:

$$a \& (b \vee c) = a \& b \vee a \& c, \; a \vee (b \& c) = (a \vee b) \& (a \vee c)$$

а також наступним аксіомам заперечення:

$$\neg\neg a = a$$

$$\neg(a \,\&\, b) = \neg a \vee \neg b,\ \neg(a \vee b) = \neg a \,\&\, \neg b$$

Приклади булевих алгебр:

1. Алгебра логіки. Нехай $X = \{x_1,...,x_n,...\}$, $n \in N$ – алфавіт, елементами якого є прості змінні з індексами (змінні виду x_j). В алгебрі логіки змінні приймають значення із множини $\{f,t\}$. (f – брехня, t – істина). Операції цієї алгебри визначені так званими таблицями істинності:

x	y	$x \,\&\, y$	$x \vee y$	$\neg x$
F	f	f	f	t
F	t	f	t	t
T	f	f	t	F
T	t	t	t	F

Виконання всіх аксіом булевої алгебри можна перевірити обчисленнями з використанням таблиць істинності. Зазначимо, що в алгебрі логіки вводиться ще кілька похідних операцій. Наприклад, операція імплікації вводиться визначенням $a \to b = \neg a \vee b$.

2. Алгебра множин. Нехай U – деяка множина, яка називається універсальною. Через 2^U позначимо множину всіх підмножин множини U. За визначенням, для $A, B \in U$

покладемо

$$A \,\&\, B = A \cap B,\ A \vee B = A \cup B,\ \neg A = U - A \ .$$

Тим самим на U визначена булева алгебра .

Детальніше з теорією булевих алгебр можна познайомитися в [16].

1.2.6. Поля

Полем F називається комутативне кільце з одиницею, множина ненульових елементів якого утворює групу. Це означає, що будь-який ненульовий елемент поля є оберненим. Інакше, будь-який елемент поля можна розділити на будь-який ненульовий елемент. Очевидно, що поле є комутативною областю цілісності. Навпаки, комутативну область цілісності можна розширити до поля.

Поле називається простим, якщо воно не містить власних підполів. Прикладом простого поля є поле Q раціональних чисел.

Характеристика поля. Найменше натуральне число k таке, що для будь-якого $a \in F$ $k \cdot a = \mathrm{df} = \underbrace{a + a + ... + a}_{k \text{ раз}} = 0$ називається характеристикою поля F.

Кажуть, що поле F має нульову характеристику, якщо ніяка сума додатної кількості однакових елементів поля не дорівнює нулю, тобто із $ka = 0$ слідує $k = 0$.

У комп'ютерній алгебрі велику роль відіграють скінченні поля, тобто поля зі скінченним числом елементів.

Приклад скінченного поля. Нехай p - просте число. Елементи скінченного поля F_p позначимо через $0,1,2,..., p-1$. $F_p = \{0,1,2,..., p-1\}$. Введемо на множині F_p операцію складання $+_p$ наступним визначенням:

$$a +_p b = (a + b)(\mathrm{mod}\, p). \quad a \cdot_p b = (a \cdot b)(\mathrm{mod}\, p).$$

див. приклад *Кільця остач* п.1.2.4 .

Можна перевірити, що множина F_p з цими операціями утворює поле. Покажемо, зокрема, що кожен елемент поля F_p, відмінний від нуля, є оберненим. Нехай $0 < a < p.$ Тоді a взаємно просте з p. У відповідності з властивістю взаємно простих чисел, для пари (a, p) існує така пара цілих чисел $(q,r), 0 < q < p$ що $aq + pr = 1.$ (Див. п. 1.1.2 Розширений алгоритм Евкліда). Це означає, що $a \cdot_p q = 1$, тобто в полі F_p $q = a^{-1}$. Можна показати, що для будь-якого p скінченне поле F_p є простим.

Розширення полів. Поле F_1 називається розширенням поля F_0, якщо $F_0 \subset F_1$. Якщо F_1 – розширення F_0, F_1 – векторний простір над F_0 (див. п.1.2.7).

Приклади полів:

1. Нехай p – просте число. Множина $Q_p = \{a + b\sqrt{p} : a, b \in Q\}$ - розширення поля раціональних чисел Q.

2. Нехай x – змінна. Множина

$$Q(x) = \{a(x)/b(x) : a(x), b(x) \in Q[x]\}$$

– розширення поля раціональних чисел Q.

Теорію полів викладено у підручниках з алгебри, книгах [17-18].

1.2.7. Векторні простори

Векторним простором V над полем F називається абелева група, на якій визначена бінарна операція $\alpha \cdot a \in V$ множення елемента $a \in V$ на елемент поля $\alpha \in F$, що задовольняє аксіомам

$1_F \cdot a = a,$

$\alpha \cdot (\beta \cdot a) = (\alpha\beta) \cdot a,$

$(\alpha+\beta)\cdot a=\alpha\cdot a+\beta\cdot a$,

$\alpha\cdot(a+b)=\alpha\cdot a+\alpha\cdot b$.

Елементи векторного простору V називають векторами, а елементи поля F – скалярами. Векторні простори також називають лінійними просторами.

Підкреслимо, що операція множення вектора на скаляр має тип $F\times V\to V$. Це – приклад операції з різнотипними аргументами.

Нехай F_0, F – поля і $F_0\subset F$. Тоді векторний простір V над F є також векторним простором V над F_0.

Приклади векторних просторів:

1. Нехай F – поле і $V=\{(\alpha,\beta,\gamma):\alpha,\beta,\gamma\in F\}$. На безлічі V визначимо операцію додавання:

$$(\alpha_1,\beta_1,\gamma_1)+(\alpha_2,\beta_2,\gamma_2)=df=((\alpha_1+\alpha_2,\beta_1+\beta_2,\gamma_1+\gamma_2))\quad (\alpha 2,\beta 2,\gamma 2)$$

Відносно цієї операції множина V утворює абелеву групу. Введемо операцію множення вектора на скаляр:

$$\delta\cdot(\alpha,\beta,\gamma)=(\delta\alpha,\delta\beta,\delta\gamma)$$

Тим самим на множині V визначено векторний простір над F. Цей простір називається тривимірним арифметичним простором над полем F.

2. Нехай F – поле, $X=\{x_1,x_2,...\}$ – нескінченний набір змінних. Визначимо множину скінченних лінійних комбінацій змінних з X із коефіцієнтами з F наступним чином:

$$V=\{\alpha_1x_{j_1}+\alpha_2x_{j_2}+...+\alpha_{kj}x_{j_{k_j}}:x_{j_1},x_{j_2},...,x_{j_{k_j}}\in X,\alpha_1,\alpha_2,...,\alpha_{kj}\in F\}$$

На V природним чином визначаються операції складання лінійних комбінацій і множення лінійної комбінації на скаляр. Тим самим на множині V визначено векторний простір над F – простір лінійних комбінацій.

Скінченна або нескінченна система (множина) ненульових векторів $a_1,.a_2..,a_n,...\in V$ називається лінійно незалежною, якщо для будь-якого n та будь-якого набору скалярів $\alpha_1,...,\alpha_n$ з того , що $\alpha_1a_1+...+\alpha_na_n=0$ слідує $\alpha_1=0,...,\alpha_n=0$. Кількість векторів (потужність множини) у максимальній лінійно незалежній системі називається розмірністю простору V. Розмірність простору V позначається через $Dim\,V$. Розмірність векторного простору V над полем F позначається через $Dim\,(V:F)$.

У прикладі 1 $Dim\,V=3$. У прикладі 2 $Dim\,V=\infty$, оскільки кожна зі змінних набору $x_1,x_2,...$ є лінійною комбінацією, тобто елементом простору V і для будь-якого натурального n елементи $x_1,x_2,...,x_n$ лінійно незалежні.

Розмірність – фундаментальна характеристика векторного простору. Якщо розмірність простору дорівнює n, будь-яка лінійно незалежна система

векторів містить рівно n векторів. У прикладі 2 розмірність простору V нескінченна, оскільки припущення про скінченність розмірності призводить до протиріччя.

Розширення полів як векторні простори. Якщо поле F_1 – розширення поля F_0, то для $\alpha \in F_0, a \in F_1$ визначена операція множення αa. Можна перевірити, що виконуються всі аксіоми множення вектора на скаляр. Тому F_1 утворює векторний простір над F_0. Якщо $Dim(F_1 : F_0) = n < \infty$, число n називається ступенем розширення F_1 над F_0 і позначається через $F_1 : F_0$.

Найменше розширення поля F, що містить елемент $a \notin F$, називається простим розширенням поля F за допомогою елемента a і позначається через $F(a)$.

У прикладі 1 п. 1.2.6 поле $Q_p = Q(\sqrt{p})$ має степінь розширення 2 над полем Q, оскільки система векторів $(1, \sqrt{p})$ – максимальна лінійно незалежна система векторів з $Q(\sqrt{p})$. Це поле можна визначити, як найменше поле, що містить і число, яке, у свою чергу, визначається як корінь алгебраїчного рівняння.

У прикладі 2 п. 1.2.6 поле $Q(x)$ також визначається як найменше поле, що містить Q і змінну x. Однак $Q(x) : Q = \infty$, оскільки максимальна лінійно незалежна система векторів $\{1, x, x^2, ..., x^n, ...\}$ є нескінченною.

Розширення скінченних полів. Теорія розширень скінченних полів грає в комп'ютерній алгебрі визначну роль. Тому наведемо основні дефініції та факти цієї теорії.

Нехай F – скінченне поле характеристики p, де p – просте число. Найменшим таким полем є поле остач за модулем p, яке позначимо через F_p.

Поле F_p є простим, оскільки будь-яке поле характеристики p містить F_p у якості підполя. Елементи F_p будемо позначати числами $0,1,2,..., p-1$.

Для будь-якого скінченного поля F кількість елементів F визначається формулою $|F| = p^d$, де d – ступінь розширення F над полем G_p характеристики p.

Розширення $F_p \subset F$ є простим, якщо існує такий незвідний поліном $g(x) \in F_p[x]$, що $F \cong G_p[x]/(g(x))$. Нехай $\deg(g) = d$. Легко побачити, що довільний елемент поля F можна представити у вигляді полінома

$$f(x) = a_0 x^{d-1} + a_1 x^{d-2} + ... + a_d,\ a_i \in G_p,\ i \in 0..d.$$

Тому поле F має p^d елементів. Число $d = \deg(g)$ називається ступенем розширення F над G_p.

Поле F є векторним простором над полем G_p, причому ступінь розширення F над F_p є розмірністю цього простору. Елементами F є поліноми ступенів менших, ніж $d = \deg(g)$ з коефіцієнтами із F_p, причому операції векторного простору виконуються над поліномами як над векторами коефіцієнтів. Множення $()\cdot_F()$ в полі F виконується за модулем $g(x)$:

$$f_1(x)\cdot_F f_2(x) = (f_1(x)\cdot f_2(x))(\mathrm{mod}\, g(x)).$$

Так само, як і для поля остач по простому модулю, можна показати, що кожний ненульовий елемент поля F є оберненим. Насправді, якщо $g(x)$ незвідний над G_p, а $\deg(f(x)) < d$, поліноми $f(x), g(x)$ взаємно прості. Тому існують такі поліноми $a(x), b(x) \in F_p[x]$, що $a(x)f(x) + b(x)g(x) = 1$. Отже, $a(x)f(x) = 1(\mathrm{mod}\, g(x))$. Це означає, що $a(x)\cdot_F f(x) = 1$, тобто $f^{-1}(x) = a(x)$.

Аналогічно визначається розширення будь-якого скінченного поля. Саме розширення $F_0 \subset F$ називається простим, якщо існує такий незвідний поліном $g_1(x) \in F_0[x]$, що $F \cong F_0[x]/(g(x))$. Поле F має $|F_0|^{\deg(g_1)}$ елементів.

Будь-яке скінченне розширення $F_p \subset H$ поля G_p є ланцюжком простих розширень виду $G_p \subset F_1 \subset F_2 \subset ... \subset F_k = H$. Якщо H – скінченне розширення поля F ступеня s, то $|H| = q^s = p^{ds}$.

Будь-яке скінченне розширення $F_p \subset H$ поля G_p є простим. Можна показати, що мультиплікативна група поля F_p є циклічною. Тому будь-який ненульовий елемент F_p є деяким ступенем твірного елемента цієї групи.

Автоморфізми полів. Автоморфізмом φ поля F називається ізоморфізм $\varphi: F \to F$, тобто таке взаємно-однозначне відображення $\varphi: F \to F$, що

$$\varphi(a+b) = \varphi(a) + \varphi(b);\ \varphi(a\cdot b) = \varphi(a)\cdot\varphi(b).$$

Нехай $F, F_0 \in F -$ розширення поля F_0. Якщо автоморфізм $\varphi: F \to F$ такий, що для кожного $a \in F_0$ $\varphi(a) = a$, кажуть, що φ залишає на місці поле F_0. Якщо φ – автоморфізм поля F, що залишає на місці поле F_0, і $f(x) \in F_0[x]$, то $\varphi(f(x)) = f(\varphi(x))$. Тому, якщо $\alpha \in F$ – один з коренів $f(x) \in F_0[x]$ і φ – автоморфізм поля F, що залишає на місці поле F_0, то $\varphi(\alpha)$ – також корінь $f(x)$. Таким чином, при автоморфізмах поля F, що залишають на місці поле F_0, множина коренів будь-якого полінома $f(x) \in F_0[x]$ переходить в себе.

Нехай $f(x) = x^s + a_1x^{s-1} + ... + a_s \in F_0[x]$ – незвідний поліном ступеня s над скінченним полем F_0 і α – один з його коренів в F. Визначимо відображення

$\varphi: F \to F$ таким чином: для довільного елемента $a \in F$ покладемо $\varphi(a) = a^q$. Покажемо, що відображення $\varphi: F \to F$ є автоморфізмом поля F. Дійсно:

$$\varphi(ab) = (ab)^q = a^q b^q = \varphi(a)\varphi(b),$$

$$\varphi(a+b) = (a+b)^q = a^q + \sum_{j=1}^{q-1} C_q^j a^j b^{q-j} + b^q = a^q + b^q = \varphi(a) + \varphi(b),$$

і усі біноміальні коефіцієнти $C_q^j,\ j = 1,...,q$ дорівнюють нулю в полі F_0. Тоді усі його корені – суть $\alpha, \varphi(\alpha), \varphi^2(\alpha)..., \varphi^{s-1}(\alpha)$. Нагадаємо, що

$$\varphi(\alpha) = \alpha^q,\ \varphi^2(\alpha) = \alpha^{q^2},...,\varphi^{s-1}(\alpha) = \alpha^{q^{s-1}}.$$

Відзначимо, що $\varphi^s(\alpha) = \alpha$. Нарешті, якщо $a \in F_0,\ a^q = a$. Насправді, мультиплікативна група поля F_0 має порядок. $q-1$ $a^{q-1} = 1$, тому $\varphi(a) = a^q = a$.

Евклідові простори. Векторний простір V над полем дійсних чисел R називається евклідовим простором, якщо на ньому введено бінарну операцію скалярного множення векторів, аргументи якої – елементи V, а результат належить R, тобто операція скалярного множення має тип $V \times V \to R$. Скалярний добуток векторів a,b позначається через (a,b). Скалярний добуток векторів задовольняє аксіомам

$(a,b) = (b,a)$,
$(\alpha a,b) = \alpha(a,b)$,
$(a,b+c) = (a,b) + (a,c)$,
$(a,a) \geq 0$,
$(a,a) = 0 \leftrightarrow a = 0$.

Нормою (довжиною) $|a|$ вектора називається величина $\sqrt{(a,a)}$:

$$|a| = df = \sqrt{(a,a)}.$$

Кут між векторами визначається формулою

$$Cos\varphi = Cos(\angle a,b) = df = \frac{(a,b)}{|a||b|}$$

Вектори a,b називаються компланарними (перпендикулярними), якщо $(a,b) = 0$. Вектори називаються колінеарними (паралельними), якщо $(a,b) = |a||b|$.

У евклідовому просторі можна вводити поняття геометричних фігур і вивчати геометрію. Наприклад, трикутником в евклідовому просторі називається така трійка векторів a,b,c, що $a+b+c = 0$.

Приклад евклідового простору: У тривимірному арифметичному векторному просторі R_3 над полем дійсних чисел R скалярний добуток векторів $a=(\alpha_1,\beta_1,\gamma_1), b=(\alpha_2,\beta_2,\gamma_2)$ визначається формулою

$$(a,b)=((\alpha_1,\beta_1,\gamma_1),(\alpha_2,\beta_2,\gamma_2))=df=\alpha_1\alpha_2+\beta_1\beta_2+\gamma_1\gamma_2.$$

тоді

$$\sqrt{|a|}=\sqrt{\alpha_1^2+\beta_1^2+\gamma_1^2}$$

Трійка векторів $e_1=(1,0,0), e_2=(0,1,0), e_3=(0,0,1)$ утворює ортонормований базис простору R_3. Це означає, що

1. $|e_1|=|e_2|=|e_3|=1$ (нормованість),
2. $(e_1,e_2)=(e_1,e_3)=(e_2,e_3)=0$ (ортогональність),
3. будь-який вектор $a=(\alpha,\beta,\gamma)\in R_3$ можна єдиним чином представити у вигляді лінійної комбінації $a=\alpha e_1+\beta e_2+\gamma e_3$.

Викладення теорії векторних просторів можна знайти у [7, 8].

1.2.8. Лінійні алгебри

Наведемо два визначення лінійної алгебри, що відрізняються тільки літературним стилем.

1. Лінійною алгеброю L над полем F називається векторний простір над полем F, на якому введено бінарну операцію множення типу $L\cdot L\to L$, відносно якої L є кільцем.

2. Лінійною алгеброю L над полем F називається кільце, на якому введено бінарну операцію множення типу $F\cdot L\to L$, відносно якої L є векторним простором.

Підкреслимо, що в лінійній алгебрі визначено два типи операцій множення.

Приклади лінійних алгебр:

1. Множина $n\times n$ матриць над полем F утворює лінійну алгебру над F.
2. Множина поліномів $F[x]$ над полем F утворює лінійну алгебру над F.
3. Множина поліномів $F[x_1,...,x_n]$ над полем F утворює лінійну алгебру над F.

1.2.9. Впорядковані множини

Множина A називається частково впорядкованою, якщо на ній задано часткове відношення $a<b$, відносно якого виконуються аксіоми строгого порядку :

$\neg(a < a)$,

$(a < b) \& (b < c) \to (a < c)$.

Відношення порядку – частково визначена бінарна операція типу $A \times A \to Bool$, де $Bool$ – булева алгебра, визначена на множині з двох елементів: $Bool = \{f, t\}$, f означає «брехня» , а t – «істина ».

Відношення нестрогого порядку задається визначенням

$$a \le b = (a = b) \vee (a < b)$$

Приклади впорядкованих множин :

1. Алгебра множин. Нехай B – множина всіх підмножин деякої множини U : $B = \{s : s \subseteq U\}$ Елементи B будемо позначати малими латинськими літерами. На B відношення строгого включення $a \subset b$ задає відношення строгого порядку.

2. Відношення подільності на множині натуральних чисел N . $a < b$, якщо a – власний дільник b .

Лінійний порядок

Порядок "<" називається лінійним, якщо виконується аксіома

$$\neg(a = b) \to (a < b) \vee (b < a)$$

Приклади:

1. Множина натуральних чисел N відносно відношення «a менше b» утворює лінійний порядок.

2. Множина раціональних чисел відносно відношення «a менше b» утворює лінійний порядок.

Щільний порядок

Лінійний порядок "<" називається щільним, якщо виконується аксіома

$$a < b \to \exists c (a < c) \& (c < b)$$

Приклади:

1. Множина раціональних чисел відносно відношення «a менше b» утворює щільний порядок.

2. Множина чисел виду $a + b\sqrt{p}, a, b \in Q, p \in \mathrm{Pr}ime$ відносно відношення «a менше b» утворює щільний порядок.

1.2.10. Відношення еквівалентності

Бінарне відношення $a \sim b$ на множині A називається відношенням еквівалентності, якщо виконуються аксіоми

$a \sim b$ – властивість рефлексивності,

$a \sim b \to b \sim a$ – властивість симетричності,

$(a \sim b)\,\&\,(b \sim c) \to (a \sim c)$ – властивість транзитивності.

Відношення еквівалентності можна інтерпретувати як бінарну операцію типу $A \times A \to Bool$.

Основна властивість еквівалентності. Відношення еквівалентності, задане на множині A, розбиває цю множину на класи еквівалентності. Дійсно, існує такий набір $\{A_j,\ j \in J\}$ підмножин множини A, що

$$A = \bigcup_{j \in J} A_j\ ,\ i \neq j \to A_i \cap A_j = \varnothing\ ,\ a \sim b \leftrightarrow \exists j (a,b \in A_j)\,.$$

Множина J – множина індексів класів еквівалентності. Зворотно, будь-яке розбиття множини A в об'єднання непересічних підмножин задає відношення еквівалентності: $(a,b \in A_j) \leftrightarrow a \sim b$.

Приклади еквівалентностей:

1. Розглянемо множину $Q = Z \times N$. Визначимо на A відношення еквівалентності: $(a,b) \sim (c,d) \leftrightarrow ad = bc.$ Перевіривши аксіоми еквівалентності, можна переконатися у тому, що задане відношення дійсно є відношенням еквівалентності. Дане відношення еквівалентності на множині пар цілих чисел задає відношення рівності звичайних дробів: $\frac{a}{b} = \frac{c}{d}$. Тому раціональним числом можна назвати клас еквівалентності на множині Q.

2. Розглянемо множину $Q[[x]]$ цілих алгебраїчних виразів від змінної x з раціональними коефіцієнтами. Цю множину також можна визначити як найменше кільце, що містить поле Q і змінну x. Визначимо на $Q[[x]]$ відношення еквівалентності таким чином: для

$$a(x), b(x) \in Q[[x]]\ a(x) \sim b(x) \leftrightarrow \forall r \in Q(a(r) = b(r))\ .$$

Іншими словами, $a(x) \sim b(x)$, якщо їх значення при будь-яких раціональних значеннях x рівні. Перевіривши аксіоми еквівалентності, можна переконатися, що дане відношення є відношенням еквівалентності .

Визначимо тепер на $Q[[x]]$ відношення еквівалентності $a(x) \approx b(x)$ наступним чином: $a(x) \approx b(x)$, якщо $a(x) - b(x)$ рівносильними перетвореннями зводиться до нуля. Під рівносильними перетвореннями розуміються розкриття дужок, спрощуючі перетворення степенів, зведення подібних, перестановки доданків і співмножників, спрощення, пов'язані з елементами 0,1 (див. підручник Алгебра – 7 клас). Зі шкільного курсу алгебри відомо, що цілий алгебраїчний вираз від $a(x)$ можна привести до вигляду

$$a(x) = \alpha_0 x^n + \alpha_1 x^{n-1} + ... + \alpha_{n-1} x + \alpha_n$$

тобто до стандартного поліноміального виду. Таким чином, $a(x) \approx b(x)$ тоді і тільки тоді, коли $a(x), b(x)$ представляють один і той же поліном. Таким чином,

$$(a(x) \approx b(x) \leftrightarrow (a(x) \overset{Q[x]}{=} b(x)).$$

Можна довести, що

$$a(x) \approx b(x) \leftrightarrow a(x) \sim b(x).$$

Конгруенції і фактор-алгебри. Нехай на множині A задана операція $y = f(x_1,...,x_n)$ і відношення еквівалентності $a \sim b$. Це відношення називається конгруенцією відносно операції f, якщо

$$a_1 \sim b_1,...,a_n \sim b_n \to f(a_1,...,a_n) \sim f(b_1,...,b_n).$$

Якщо відношення еквівалентності $a \sim b$ – конгруенція відносно будь-якої операції даної алгебри **A**, воно називається конгруенцією на **A**. Конгруенції – це еквівалентності, що зберігаються при алгебраїчних операціях.

Нехай $a \sim b$ – конгруенція на алгебрі А. У кожному класі A_j еквівалентності, заданому цією конгруенцією, виберемо один елемент $a_j \in A_j$. Цей елемент представляє клас еквівалентності. Розглянемо множину $A_\sim = \{a_j : j \in J\}$ Тепер на $A_\sim$ можна коректно визначити всі операції алгебри А. Дійсно, якщо f_A – операція на A, а $f_{A_\sim}$ – та ж операція на $A_\sim$, то $f_{A_\sim}(a_1,...,a_n) = f_A(a_1,...,a_n)$. Алгебра $A_\sim$ називається фактор-алгеброю алгебри A по конгруенції $a \sim b$. Ця фактор-алгебра позначається через $A/\sim$.

Еквівалентності, розглянуті в прикладах 1, 2 цього пункту, є конгруенціями. Отже, ці алгебри можна факторизувати.

Приклади конгруенцій:

3. (продовження прикладу 1). У класі еквівалентності, якому належить пара (a,b), виберемо в якості представника таку пару (a',b'), що $\gcd(a',b') = 1$. Можна переконатися у тому, що фактор-алгебра $Q[[x]]/\sim$ складається з нескоротних дробів з натуральними знаменниками. $Q/\sim$ називають полем раціональних чисел і позначають через Q (Quotients).

4. (продовження прикладу 2). У класі еквівалентності, якому належить $a(x)$, виберемо в якості представника поліном стандартного виду $a(x) = \alpha_0 x^n + \alpha_1 x^{n-1} + ... + \alpha_{n-1} x + \alpha_n$. Можна переконатися в тому, що фактор-алгебра $Q[[x]]/\sim$ складається з поліномів стандартного виду. Цю алгебру називають кільцем поліномів від x над полем Q і позначають через $Q[x]$.

Матеріал з теорії відношень можна знайти у підручниках [4, 7, 8].

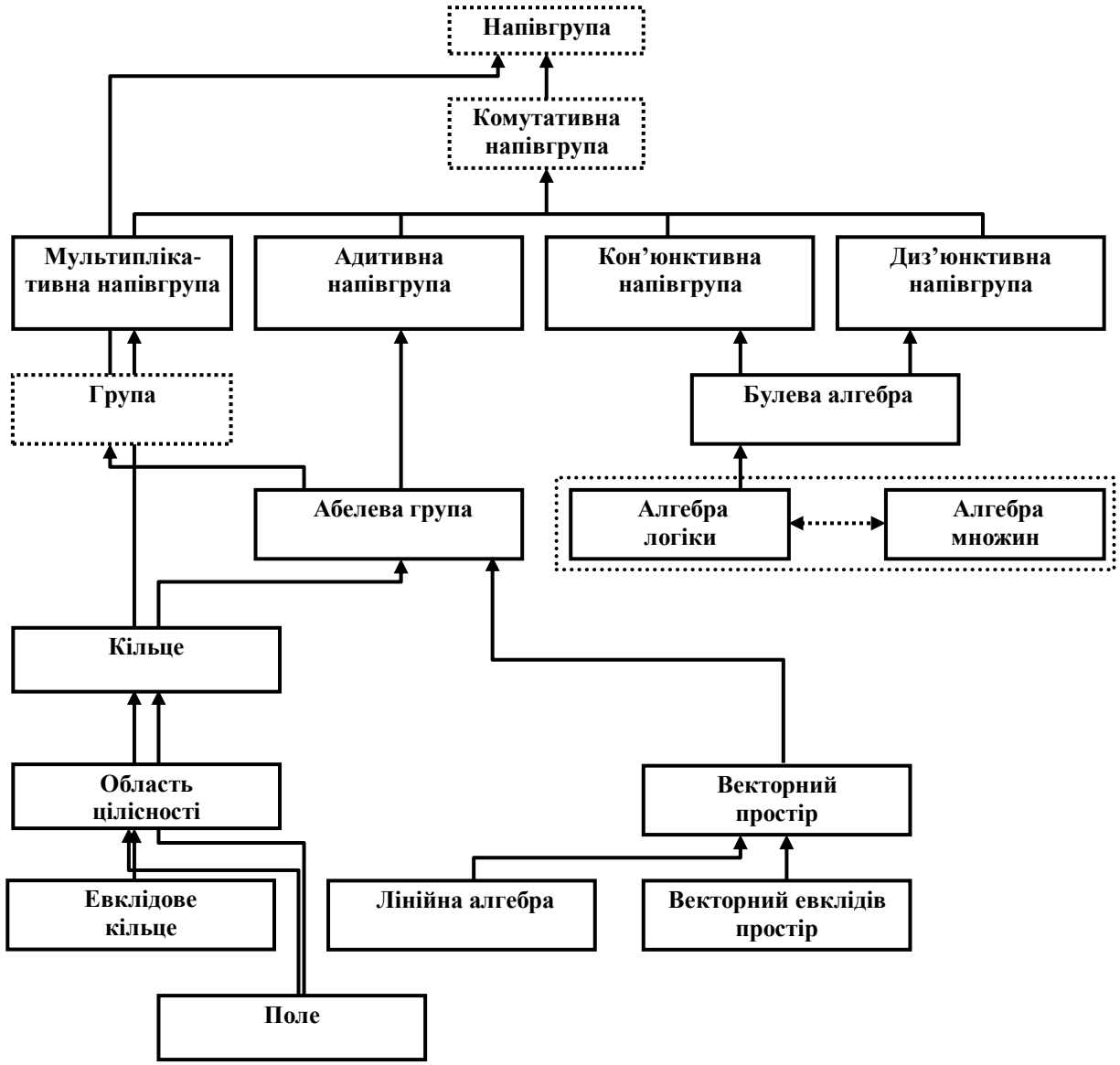


Рис.1.2. Ієрархія наслідування алгебр розділу 1.2.

1.3. Висновки: визначення класичних алгебр

1. Визначення класичних алгебр, наведені вище, використовують відношення «Алгебра *A* є алгеброю *B*». Це відношення у об'єктно-орієнтованому програмуванні називають відношенням наслідування.

Сукупність алгебр, пов'язаних відношенням наслідування, утворює ієрархію наслідування (Рис. 1.2)

2. Визначення класичних алгебр використовують аксіоми, що задають їх абстрактні властивості, тобто властивості, які не використовують представлені елементи носіїв алгебр. Разом з тим в прикладах без конструктивних визначень «зовнішнього вигляду» елементів алгебри обійтися не можна. Комп'ютерна алгебра має справу з алгебрами, заданими конструктивно.

3. У визначеннях алгебр використовуються типи алгебраїчних операцій, які визначаються типами аргументів і результатів операцій . Ці типи можуть бути різними для різних операцій, але вони об'єднують визначення алгебр в єдине ціле. Так, векторний простір використовує поняття поля (скалярів). Разом з тим поняття розширення поля використовує поняття векторного простору.

Глава 2. Технології алгебраїчного програмування. Система програмування АПС

У багатьох предметних областях побудова розв’язання прикладної задачі вимагає перетворень математичних об'єктів. Типова ситуація полягає в наступному: математична модель об'єкта задається у вигляді конструктивного математичного об'єкта, а предметну область задано переліком припустимих елементарних перетворень математичних об'єктів. Розв’язання прикладної задачі полягає у тому, щоб, послідовно перетворюючи об'єкт припустимими перетвореннями, привести його до необхідного (простого) виду - відповіді.

Наприклад, математичною моделлю об'єкта є система лінійних рівнянь, конструктивно задана розширеною матрицею системи. Припустимі перетворення – суть елементарні перетворення рядків матриці. У задачі потрібно знайти точний розв’язок системи. Задачі такого роду природно розв’язувати методами алгебраїчного програмування. Найбільш адекватними системами алгебраїчного програмування виявилися системи, що використовують технології переписувань [19 - 21].

2.1. Рівності як правила переписування

Відношення рівності в математиці використовується у формулюваннях тотожностей, співвідношень в означеннях, рівнянь і т. ін. В алгебраїчному програмуванні знак рівності використовується у так званих правилах переписування для позначення припустимого перетворення математичного об'єкта.

Приклад 2.1.
Правило переписування: `x + 0 = x`
Математичний об'єкт: `2*A + 0`
Результат переписування: `2*A`.

Приклад 2.2.
Правило переписування**:** `(a*x = b) = (x = b/a)`
Математичний об'єкт: `2*y = a + 4`
Результат переписування: `y = (a + 4)/2`.

Правило переписування може бути застосованим до одних об'єктів і не застосованим до інших об'єктів. У випадку, коли правило не застосоване до об'єкта, у результаті його дії об'єкт не перетвориться. Таким чином, незастосовність правила інтерпретується як тотожне перетворення об'єкта «на місці». Відзначимо, що в обох випадках на застосування правила переписування витрачається комп’ютерний час.

Приклад 2.3.
Правило переписування: `x + 0 = x`
Математичний об'єкт: `0 + 2*A`
Результат переписування: `0 + 2*A`.

Приклад 2.4.
Правило переписування: `(a*x = b) = (x = b/a)`
Математичний об'єкт: `-y = 4`
Результат переписування: `-y = 4`.

У системах програмування, що використовують технології переписувань, перетворення, визначене правилом переписування, виконується у два етапи: спочатку ліва частина правила переписування (зразок) синтаксично співставляється (порівнюється) із перетворюваним об'єктом. Якщо синтаксична структура об'єкта збігається із синтаксичною структурою зразка, виконується перетворення об'єкта, визначене його правою частиною.

Процедура співставлення об'єкта зі зразком називається метчінгом (matching). Метчінг не тільки співставляє об'єкт зі зразком, але й знаходить фактичні значення змінних, ідентифікатори яких зазначені у зразку. Ці значення є підвиразами перетворюваного об'єкта.

Приклад 2.5. (продовження приклада 2.1).
Правило переписування**:** `x + 0 = x`
Зразок: `x + 0`
Математичний об'єкт: `2*A + 0`
Результат порівняння додатний: `2*A + 0` має вигляд `x + 0`
Фактичне значення `x:` `2*A`
Результат переписування: `2*A`.

Приклад 2.6. (продовження приклада 2.2).
Правило переписування: `(a*x = b) = (x = b/a)`
Зразок: `a*x = b`
Результат порівняння додатний: `2*y = a+4` має вигляд `a*x = b`
Фактичні значення параметрів: `a: 2, x: y, b: a + 4.`
Математичний об'єкт: `2*y = a + 4`
Результат переписування: `y = (a + 4)/2`.

Для визначення зразка – лівої частини правила переписування, крім змінних, які грають роль формальних параметрів, використовуються ще й константи, що позначають самі себе. У прикладі 2.5 такою константою є **0.** Оскільки позначення констант потрібно відрізняти від позначення змінних – формальних параметрів, правило переписування потрібно доповнити відповідною інформацією.

Приклад 2.7.
Правило переписування:
x: змінна, `True`: константа
`x & True = x`
Математичний об'єкт: `(A ∨ B) & True`

Результат: (A ∨ B).

Іноді зразок – ліва частина правила переписування містить кілька входжень однієї й тієї ж змінної. У цьому випадку відповідні підвирази – фактичні параметри перетворюваного виразу співставляються синтаксично.

Приклад 2.8.
Правило переписування**:** x&y ∨ x&z = x&(y ∨ z)
1. Математичний об'єкт: A&B ∨ A&C
Результат: A&(B ∨ C)
2. Математичний об'єкт: (A∨B)&C ∨ (B∨A)&D

Результат: (A∨B)&C∨(B∨A)&D // правило незастосовне: A∨B не співставлено синтаксично з B∨A.

Поряд з безумовними правилами переписування, в алгебраїчному програмуванні використовуються й умовні правила переписування.

Для позначення мовної конструкції Якщо A то B використовується стрілка (знак імплікації): A→B. Наприклад, умовне правило переписування, що визначає множення обох частин нерівності на додатне число, запишеться у вигляді

(c > 0) → (A < B = c*A < c*B).

Приклад 2.9.
Правило переписування**:** (a>0) → (a*x < b = x < b/a)
1. Математичний об'єкт: 5*y < 3 + x
Результат: y < (3+x)/5
2. Математичний об'єкт: -5*y < 3 + x
Результат: -5*y < 3+x // правило незастосовне.

Приклад 2.10.
Правило переписування: (GCD(a,b) == 1) -> (a/b = a//b)
Математичний об'єкт: 7/12
Результат: y < 7//12.

Таким чином, при спробі застосування умовного правила переписування спочатку обчислюється логічне значення умови. Якщо умова приймає значення True, виконується метчінг, визначення фактичних параметрів, співставлення на рівність виразів, що відповідають різним входженням того самого формального параметра, і, нарешті, переписування.

2.2. Системи правил переписувань

Для того, щоб технології переписування стали ефективним інструментом алгебраїчного програмування, окремі правила переписування потрібно з'єднувати у так звані системи правил переписувань (rewriting rules systems).

Приклад 2.11. Система правил переписувань, що визначає властивості 0 і 1 для спрощення алгебраїчних виразів.

```
( x*1 = x,
 1*x = x,
 x + 0 = x,
 0 + x = x,
 x*0 = 0,
 0*x = 0);
```

Застосувати систему правил переписувань до математичного об'єкта означає застосувати таке правило цієї системи, що призведе до результату. Більш точно, застосування системи переписувань до об'єкта полягає у повторенні спроб застосування окремих правил цієї системи в порядку «зверху – вниз» їхнього розташування в системі, починаючи з першого:

```
( x*1 = x,          (1)
 1*x = x,           (2)
 x + 0 = x,         (3)
 0 + x = x,         (4)
 x*0 = 0,           (5)
 0*x = 0);          (6)
```

Повторення закінчується у випадку, коли:

1. Одне із правил системи застосовано до даного об'єкта. Результатом застосування системи до даного об'єкта є результат застосування цього правила до об'єкта.

2. Жодне з правил системи незастосовне. Результатом застосування системи до даного об'єкта у цьому випадку є даний об'єкт.

Приклад 2.12. Правила множення ступенів з однією основою, оформлені у вигляді системи правил переписувань.

```
( x^m * x^n = x^(m + n),
 x * x^n = x^(1 + n),
 x^m * x = x^(m + 1),
 x*x = x^2);
```

Відзначимо, що перше із цих правил описує загальну ситуацію, а інші - ті частинні її випадки, у яких зразок змінюється синтаксично.

Семантика системи переписувань залежить, взагалі кажучи, від порядку розташування правил в системі. Наприклад, системи

```
(a+b = b+a, a+0 = a)  і (a+0 = a, a+b = b+a),
```

що відрізняються тільки порядками розташування правил, застосовані до виразу `x+0`, призведуть до різних результатів.

2.3. Стратегії переписувань

Однієї тільки можливості об'єднання правил переписування у системи недостатньо для того, щоб зробити технологію переписувань ефективним інструментом алгебраїчного програмування. Справді, система переписувань прикладу 2.11 містить усі співвідношення, що спрощують вираз за властивостями нуля та одиниці. Однак для того, щоб спростити вираз, цю систему потрібно застосувати неодноразово, причому не тільки до всього виразу, але й до усіх його підвиразів.

Приклад 2.13.
До усього виразу

```
x*y*1 + 0*x*y + 0
```
(2.1)

система переписувань прикладу 2.11 незастосовна, однак, якщо застосувати її кілька разів до підвиразів, результат буде досягнутий.

Для того, щоб оцінити проблему, розставимо правильно дужки у виразі (2.1):

```
x*(y*1) + (0*(x*y) + 0)
```
(2.2)

Розміщення дужок виявило структуру виразу, явно виділивши головні знаки й операнди усього виразу та його підвиразів. До усього виразу (2.2) система переписувань незастосовна. Вона застосовна безпосередньо до:

- другого операнда;
- першого операнда другого операнда;
- другого операнда першого операнда.

Після того, як зазначені перетворення будуть виконані, вираз (2.2) спроститься до виду

```
x*y + 0
```
(2.3)

і систему переписувань можна застосувати до усього виразу, отримавши

```
x*y.
```
(2.4)

У перетвореннях виразів можна виділити два способи визначення порядку обчислень: “зверху-вниз” і “знизу-вгору”. При обчисленнях “зверху-вниз” система переписувань спочатку застосовується до всього виразу, а потім – до усіх його операндів (аргументів). При обчисленнях “знизу-угору” система переписувань спочатку застосовується до усіх його аргументів, а потім – до усього виразу.

Відзначимо, що у прикладі 2.13 обчислення “зверху-униз” не дадуть правильного результату, зупинившись на виразі (2.3). Обчислення “знизу-угору” дадуть результат (2.4). І у тому, і у іншому випадках істотну роль грає порядок застосування системи переписувань до даного виразу. А саме, важливу роль грає той факт, чи застосовується система повторно до даного виразу, якщо попереднє застосування пройшло успішно?

Наприклад, повторне застосування системи до виразу `(1*a*1+ 0)+0` призведе до результату `a`.

Правило застосування системи переписувань до вираження називають стратегією переписувань.

Позначимо через S систему переписувань, а через E – алгебраїчний вираз. Перелічимо ті стратегії переписувань, які ми вже розглянули:

1. Застосувати один раз S до E.
2. Застосовувати повторно S до E (доти, поки S застосовна).
3. Застосувати стратегію 1 із S до E та його підвиразів зверху-униз.
4. Застосувати стратегію 2 із S до E та його підвиразів зверху-униз.
5. Застосувати стратегію 1 із S до E та його підвиразів знизу-угору.
6. Застосувати стратегію 2 із S до E та його підвиразів знизу-угору.

Зрозуміло, що цей список можна продовжити:

7. Застосовувати повторно стратегію 4 із S до E.
8. Застосовувати повторно стратегію 6 із S до E.

Відзначимо наступні аспекти:

1. Будь-яка стратегія переписувань має два аргументи: систему правил переписувань S і вираз, що переписується.

2. Більш складні стратегії використовують більш прості стратегії. Так, базовими (елементарними) стратегіями нашого списку стратегій є такі стратегії:

- застосувати один раз;
- застосовувати повторно;
- застосовувати зверху-униз;
- застосовувати знизу-угору.

Стратегія 8 визначається комбінацією базових стратегій:

Застосовувати_повторно
(Застосовувати_ знизу-вгору
(Застосовувати_повторно S до E))

Зрозуміло, список стратегій переписувань, наведений нами вище, не є повним. Крім того, не всі зі стратегій, описаних нами, реалізовані у системі програмування АПС, яку буде розглянуто нижче. Однак ефективність алгебраїчних програм залежить не тільки від систем правил переписувань, але й від стратегій переписувань. Тому у системі АПС, поряд із “вбудованими” стратегіями переписування, застосовується механізм визначення “своїх” стратегій, тобто стратегій користувача.

Приклад 2.14. Додавання чисел.
Висхідний вираз має вигляд $E = a_1+a_2+\ldots+a_{n-1}+a_n$, де a_i – числа.

Потрібно визначити систему правил переписування і стратегію переписувань, що додає усі ці числа. Скласти два числа `a` і `b` можна функцією `Add(a, b)`.

Розв'язок 1.

Система правил переписування S:

```
((a, b, B) = (Add(a, b), B),        // правило 1
 (a, b) = Add(a, b) );              // правило 2
```

Стратегія: Застосовувати_ повторно S до E.

Вхід: `1 + 2 + 3 + 4 + 5`.

Трасування застосувань стратегії до входу. Виділений жирним знак – головний знак виразу – "точка" застосування системи переписувань.

```
1 + (2 + (3 + (4 + 5)))   // застосувалося правило 1.
3 + (3 + (4 + 5))         // застосувалося правило 1
6 + (4 + 5)               // застосувалося правило 1
10 + 5                    // застосувалося правило 2
15/                       / жодне з правил незастосовне
```

Розв'язок 2.

Система правил переписування S:

```
((a, b) = Add(a, b));               // правило 1
```

Стратегія 6:Застосовувати_ знизу-вгору(Застосовувати_ повторно S до E).

Вхід: `1 + 2 + 3 + 4 + 5`

Трасування застосувань стратегії до входу:

```
1 + (2 + (3 + (4 + 5)))  //застосовуємо S до другого операнду.
1 + (2 + (3 + (4 + 5)))  // застосовуємо S до другого операнду.
1 + (2 + (3 + (4 + 5)))  // застосовуємо S до другого операнду.
1 + (2 + (3 + (4 + 5)))  // застосувалося правило 1.
1 + (2 + (3 + 9))        // застосувалося повторно правило 1.
1 + (2 + 12)             // застосувалося повторно правило 1.
1 + 14                   // застосувалося повторно правило 1.
15/                      / система незастосовна.
```

Приклад 2.15 є прикладом найпростішої алгебраїчної програми. Для розгляду подальших прикладів наведемо деякі синтаксичні правила оформлення таких програм.

2.4. Найпростіші алгебраїчні програми

Дані. Даними алгебраїчних програм є алгебраїчні вирази. Як відомо, атомарними виразами є константи та змінні (атоми або атомарні вирази). Для опису більш складних виразів використовуються знаки операцій і функцій, а

також дужки. Ми будемо поки використовувати будь-які операції та функції, які відомі з математики.

Приклади алгебраїчних виразів:

```
Sin(2*x+Pi/2) + Cos(2*x-Pi/2), Sqrt(Sqr(x-a) + Sqr(y-b))
```

Ми будемо також вважати знак коми “ **,** ” знаком бінарної операції з найнижчим пріоритетом. Таким чином, рядок

```
a, b+2, c*2, b^2
```

– це алгебраїчний вираз, в якому кома позначає бінарну операцію, а дужки розставлені згрупованими вправо. `a,(b+2,(c*2,b^2))`.

Імена. Імена використовуються для позначення даних, змінних, функцій, систем правил переписування. Іменами є рядки, складені з букв латинського алфавіту, цифр, а також знака підкреслення. Ім'я не може починатися із цифри. Приклади імен:

```
A, b1, b2, b3, Sin, Cos, Expression, Reduction_Rule.
```

Для того, щоб зв'язати деяке ім'я з виразом, ми будемо використовувати оператор `Set`. Оператор `Set` визначений у такий спосіб:

```
Set(<Ім'я змінної>, <Вираз>)
```

Його ж можна використовувати й у такому синтаксисі:

```
<Ім'я змінної> → <Вираз>)
```

Наприклад:

```
Set(A,Sin(2*x+Pi/2)+Cos(2*x-Pi/2))   або
   a → Sin(2*x+Pi/2)+Cos(2*x-Pi/2);
Set(S, Sqrt(Sqr(x-a) + Sqr(y-b)))   або
    S → Sqrt(Sqr(x-a) + Sqr(y-b)).
```

Синтаксис оформлення системи правил переписування (СПП)

Системи правил переписування оформлюються в такий спосіб:

```
<Ім'я СПП> := rs(<список формальних параметрів через “,”>)
                (<список правил переписування через “,”>);
```

Приклад 2.16.

```
Simplify := rs(x)(
  x*1 = x,
  1*x = x,
  x + 0 = x,
  0 + x = x,
  x*0 = 0,
  0*x = x);
```

```
Summa := rs(a, b, B) (
 (a, b, B) = (Add(a, b), B),          // правило 1
 (a, b) = Add(a, b) );                // правило 2

TrigRedRules := rs(x)(
  Sin(x + Pi/2) =  Cos(x),
  Cos(x + Pi/2) = -Sin(x),
  Sin(x + Pi)   = -Sin(x),
  Cos(x + Pi)   = -Cos(x),
  Sin(x + 2*Pi) =  Sin(x),
  Cos(x + 2*Pi) =  Cos(x));
```

Стратегії. Кожна зі стратегій переписувань, яка визначена у мові алгебраїчного програмування, має свій ідентифікатор. Цей ідентифікатор є ідентифікатором процедури, параметрами якої є ідентифікатор перетворюваного виразу та ідентифікатор системи правил переписувань. Перетворення здійснюються «на місці»: перетворений вираз позначено тим же ідентифікатором, що й вихідний вираз. У прикладах алгебраїчних програм ми будемо використовувати наступні стратегії:

`applr` – застосувати однократно;
`appls` – застосовувати повторно доти, поки застосовується;
`ntb` – застосувати зверху-вниз, застосовуючи повторно до кожного підвиразу (`appls`);
`nbt` – застосувати знизу-вгору, застосовуючи повторно до кожного підвиразу (`appls`).

Синтаксис опису процедури застосування стратегії:

<Ім'я стратегії>(<Ім'я виразу>, <Ім'я системи переписувань>)

Наприклад:

```
nbt(S, Summa);
appls(A, Simplify).
```

Приклад 2.19 Алгебраїчна програма обчислення НСД двох натуральних чисел.

`/*` Трьома точками виділена процедурна частина програми. `*/`

```
NAMES rsGCD, T;
rsGCD := rs(a, b)(
            (a, a) = a,
(a < b)->   ((a, b)=(a, b-a)),
            ((a, b)=(a-b, b))              /*a > b*/
);
. . .
T --> (48, 124);  /* Установити змінну T на пару чисел */
```

```
appls(T, rsGCD); /* Застосовувати повторно систему rsGCD до T */
print(T)         /* Роздрукувати результат T */
. . .
```

Текст нашого прикладу ілюструє структуру найпростішої алгебраїчної програми. Вона містить:

- опис змінних;
- опис систем правил переписувань;
- процедурну частину, що ініціює обчислення. Тут установлені вихідні дані, описані обчислення й виведення результату.

Для коментарів використовуються дужки `/*, */`.

Опишемо синтаксис процедурної частини алгебраїчної програми, що ініціює обчислення, мовою АПЛАН системи АПС. Виконання алгебраїчної програми починається з виконання оператора присвоєння (оператор `let`), ліва частина якого – стандартний ідентифікатор `task`**,** а права частина – оператор, що виконується. Найчастіше цей оператор є складеним. Круглі дужки в цьому випадку відіграють роль операторних дужок.

`task:=(`<список операторів через кому або точку з комою>`);`

Таким чином, програма прикладу 2.19 виглядає так:

```
NAMES rsGCD, T;
rsGCD := rs(a, b)(
              (a, a) = a,
     (a < b)->((a, b)=(a, b-a)),
              ((a, b)=(a-b, b))          /* a > b */
);
task := (T --> (48, 124); /* Встановити змінну T на пару чисел */
appls(T, rsGCD); /*Застосовувати повторно систему rsGCD до T*/
print(T)          /* Роздрукувати результат T */
);
```

Приклад 2.18. Алгебраїчна програма обчислення `n`-того числа Фібоначчі.

Вхід: натуральне число `n` – номер числа Фібоначчі.

Вихід: натуральне число `m` – число Фібоначчі з номером n.

Співвідношення, що визначають послідовність Фібоначчі:

```
F(0) = 1, F(1) = 1,

(n > 1) -> (F(n) = F(n-1) + F(n-2)).
```

Ці співвідношення можуть бути системою правил переписування у наступній найпростішій алгебраїчній програмі:

```
NAMES rsFib, T;
rsFib := rs(n)(
 F(0) = 1,  F(1) = 1,
```

```
 (n > 1) -> (F(n)= F(n-1) + F(n-2))
);
task := (T --> F(5); /* Установити змінну T на F(n) */
ntb(T, rsFib);  /* Застосувати зверху-униз rsFib до T */ print(T)
      /* Роздрукувати результат T */
);
```

Відзначимо, що ідентифікатор `F` не описаний. Отже, це – функціональна константа. Зауважимо однак, що ця програма не є ефективною. Справді, якщо здійснити трасування, можна побачити, що програма по суті розписує `F(5)` у суму одиниць, підраховуючи по ходу результат. Більш ефективний алгоритм можна отримати, якщо зводити подібні у виразі – результаті застосування загального правила обчислення числа Фібоначчі. Тоді перетворюваний вираз буде мати вигляд `a*F(n) + b*F(n-1)`, і в результаті застосування загального правила він перепишеться у виді

```
(a+b)*F(n-1) + a*F(n-1).
```

```
NAME T, rsFib;
rsFib := rs(a, b, x, y)(
  F(0) = 1,
  F(1) = 1,
  F(x) = F(x-1) + F(x-2),
  2*F(1) + F(0) = 3,
  F(1) + F(0) = 2,
  a*F(1) + b*F(0) = a + b,
  (x == y+1)-> (2*F(x) + F(y) = 3*F(y) + 2*F(y-1)),
  (x == y+1)-> (F(x) + F(y) = 2*F(y) + F(y-1)),
  (x == y+1)-> (a*F(x) + b*F(y) = (a+b)*F(y) + a*F(y-1))
);
task :=( T --> F(20);
     appls(T, rsFib);
     print(T)
    );
```

У цій версії алгоритму ми маємо справу з вектором `(a, b)` коефіцієнтів при числах `F(n), F(n-1)`. Тому загальне перетворення можна сформулювати так:

```
(a, b, n) = ((a+b), a, n-1).
```

Алгоритм перетворився у стандартний ефективний алгоритм обчислення чисел Фібоначчі:

```
NAME T, rsFib;
rsFib := rs(a, b, n)(
 F(0) = 1,
 F(n) = (1, 1, n-1),
 (a, b, 0) = a,
 (a, b, n) = (a + b, a, n-1)
```

```
);
task :=(T --> F(20);
   appls(T, rsFib);
   print(T)
  );
```

2.5. Мова алгебраїчного програмування АПЛАН

У попередньому пункті ми розглянули початкові відомості про мову АПЛАН – мову програмування високого рівня системи алгебраїчного програмування АПС [22 - 24] . Зараз ми продовжимо цей розгляд.

2.5.1. Модульна структура мови АПЛАН

Алгебраїчна програма, взагалі кажучи, складається з декількох модулів, що “збираються” під час препроцесування командою `INCLUDE`. Файли алгебраїчних модулів мають розширення **.**`ap`. Синтаксичне визначення команди `INCLUDE`:

`INCLUDE` <ім'я ap-модуля>

Команду `INCLUDE` треба розташовувати у окремому рядку. При цьому, як правило, усі такі команди групуються в перших рядках `ap`-модуля.

Для того, щоб алгебраїчна програма стала працездатною, до неї потрібно підключити стандартний `ap`-модуль `std.ap`, який містить означення стандартної множини операцій (арифметичних і логічних), арифметичних відносин, стандартних імен, і таке інше. Зокрема, приклади пункта 4 потрібно доповнити: у першому рядку модуля потрібно підключити модуль `std.ap`. Алгебраїчна програма приклада 2.19 має виглядати таким чином:

```
INCLUDE <std.ap>  /*підключення стандартного модуля std.ap */
NAMES rsGCD, T;
rsGCD := rs(a, b)(
       (a, a)  = a,
(a < b) -> ((a, b) = (a, b - a)),
       ((a, b) = (a - b, b))
);
task := (T -i> (48, 124);
appls(T, rsGCD);
print(T)
);
```

Оголошення імен. Імена (ідентифікатори) в АПЛАН використовуються для позначення змінних, систем правил переписування, процедур та функцій. Синтаксис:

```
NAME <ім'я або список імен через кому>;
NAMES <ім'я або список імен через кому>;
```

Зрозуміло, що службове слово NAME пропонується використовувати для оголошення одного імені, а NAMES – для оголошення списку імен. Однак цього правила можна і не дотримуватися, тому що обидва цих службових слова є синонімами.

Всі змінні, оголошені в алгебраїчному модулі, є глобальними. Тому їх потрібно розташовувати поза описом процедур і функцій. Механізм використання локальних змінних буде описаний пізніше.

Областю видимості змінних, описаних у даному ap-модулі, є цей модуль. При цьому, зазвичай, оголошення імен має передувати їхньому використанню.

```
 . . .
NAME rsSqEqu;
rsSqEqu := rs(a, b, c)(
 (a*x^2 = b) = ( x = Sqrt(b/a)),
 (a*x = b) = (x = b/a)
);
NAMES A, B, C;
A -i> B + C;
 . . .
```

Алгебраїчні вирази. Поняття алгебраїчного виразу в АПС є основним та універсальним. Програма, написана мовою АПЛАН, є алгебраїчним виразом. По суті, виконання алгебраїчної програми є обчисленням її значення як алгебраїчного виразу. Стандартні операції мови АПЛАН визначені у файлі std.ap. Для кожної стандартної операції засобами АПС реалізований її інтерпретатор – функція, яка обчислює значення відповідної операції, виходячи із значень її аргументів. Операція в АПС визначається як відмітка (mark) вершини так званого клубка – структури даних типу “помічене дерево”, яка містить як власно алгебраїчні програми, так і дані, які опрацьовуються програмою. Об’ява відмітки операції здійснюється у розділі відміток АПЛАН-програми, який починається службовими словами MARK, MARKS. Нижче наведено фрагмент опису стандартних операцій з файлу std.ap.

```
MARKS
/* Arithmetical and algebraic operations and functions */
POW( 2, 60, "^",0),        /* power x ^ y       */
MOD( 2, 59,"mod",0),       /* residual x mod y  */
Mult( 2, 57, "*"),         /* multiplication x * y */
DIV( 2, 58, "/",0),        /* division x / y     */
MLT( 2, 56, "$"),          /* uninterpreted      */
SUB( 2, 55, "-",1),        /* subtraction x - y  */
ADD( 2, 54, "+"),          /* addition  x + y    */
NOD( 2, 53,"nod"),         /* residual x nod y   */
```

```
/* Predicates */
LE ( 2, 40, "<=",0),        /* less or equal x <= y       */
LS ( 2, 40, "<",0),         /* less x < y                 */
ME ( 2, 40, ">=",0),        /* more or equal x >= y       */
MR ( 2, 40, ">",0),         /* more x > y                 */
EQ  ( 2, 11, "=="),         /* equality of numbers x == y */
EQU ( 2, 11, "=" ),         /* uninterpreted              */
/* Logical connections */
~ (1, 30),                  /* logical negation ~(x)      */
AND( 2, 29, "&"),           /* logical and x & y          */
OR ( 2, 28, "|/"),          /* logical or x || y          */
IFF( 2, 27, "<=>"),         /* logical equivalence x <=> y */
/* L2B operations */
SET(2, 20, "-->" ),         /* set statement x-->y        */
ASS(2, 20, ":="  ),         /* assignement statement x:=y */
ELSE(2,19,"else"),          /*used in conditional statement: x->y else z */
IF(2, 18, "->"  ),          /* implication x -> y and separator
                               in conditional statements  */
do1),                       /* do statement: do(p)        */
while(2),                   /* while statement: while(x,y) */
/* Separators */
comma( 2, 7, ","),
LL (2, 5, ";"),
/* Special functions */
(1),                /* `(x) = x */
arg(2),             /* arg(f(x1,...,xn)) = xi */
ART (1),            /* ART(x) = arity of x    */
CAN (1),            /* Basic canonical form */
IFTH(3),            /* IFTH(x,y,z) = if x then y else z  */
vl  (1),            /* vl(x) = copy of value of name x   */
VL  (1),            /* VL(x) = substitution of values to term x  */
copy(1);            /* copy(x) is copy of x if x has no loops */
```

Інтерпретатори стандартних операцій, що об'явлені у `std.ap`, реалізовані на нижньому рівні системи АПС у вигляді С-функцій (системна бібліотека `intlib.dll`). Системна таблиця `can_tbl.tbl` визначає відповідність імен операцій та їх інтерпретаторів. Фрагмент цієї таблиці, що

стосується арифметичних та алгебраїчних операцій, предикатів та логічних зв'язок, представлений нижче.

Имя отм	**инфикс форма**	**интерпрер С++ на АПЛАН**	**имя функции С++**
ADD	add	add_can	intlib
AND	and_can	and_can	intlib
ART	---	ART_can	intlib
DIV	div	div_can	intlib
EQ	eqq	Eq_can	intlib
IFF	---	iff_can	intlib
Mult	mult	mult_can	intlib
ME	---	Me_can	intlib
MOD	MD	mod_can	intlib
MR	mor	Mr_can	intlib
LE	---	Le_can	intlib
LS	low	Ls_can	intlib
NOD	ND	nod_can	intlib
OR	or_can	Or_can	intlib
POW	pow	pow_can	intlib
SUB	sub1	sub_can	intlib

Поряд зі стандартними операціями, програміст має змогу визначати власні операції та реалізовувати їх інтерпретатори. По суті, програмування в АПС можна схарактеризувати як процес визначення своїх операції та реалізацію їх інтерпретаторів. Визначення власної операції здійснюється у розділі MARK відповідно до синтаксису

```
<id операції>(<арність>,<пріоритет>{,<"інфіксне
     позначення">}{,<спец.відмітка>})
```

Наприклад:

```
Mark
      Quot(2, 52, "//", 0),
      Sin(1),
      LF(3),
      List(UNDEF);
```

Значення арності UNDEF означає операцію довільної арності. Інші значення арності мають бути невід'ємними цілими числами. Звичайно, інфіксна форма запису операції можлива лише для бінарних операцій. В інших

випадках застосовують функціональну (префіксну) форму запису операцій. Наприклад:

```
4//8, Quot(1,2), Sin(Pi), LF(x,A,B), List(1,2,3,5);
List(2009, 2010,2011,2012).
```

2.5.2. Імперативні засоби мови АПЛАН

Базові оператори. Базові оператори імперативних засобів – оператор `SET` (оператор установки) та `ASS` (оператор присвоєння) зв'язують значення алгебраїчного виразу – правої частини з іменем лівої частини. Наведемо синтаксичні означення базових операторів з [25]:

```
<basic statement>::=<set statement>|<assignment statement>
<set statement>::=<selector> --> <algebraic expression>
<assignment statement>::=<name>:=<algebraic expression>
<selector> ::= <name>
```

Для запису цих операторів зручно використовувати інфіксну форму запису

```
X --> E,   X := E.
```

Умовний оператор. Умовний оператор реалізовано операторами `IF`, `ELSE`**.** Запис скороченого і повного умовного операторів зручно здійснювати у формі

```
U -> S,    U -> S1 ELSE S2.
```

Оператор циклу. Оператор циклу `WHILE` записують у вигляді

```
WHILE(U, S)
```

Програміст має змогу засобами мови визначати власні оператори циклів. Наприклад, у системі правил переписувань `Compile` визначення циклів здійснюються по суті як скорочення

```
dowhile(x,y) = (x, while(y,x)),
for(x,y,z,u) = (x, while(y,(u,z))),
loop(x)      = while(1,x).
```

Ця система правил переписувань призначена для препроцесування алгебраїчної програми. Вона описана у файлі `gen_obj.ap` і виконується кожного разу перед виконанням АПС програми, якщо цей файл підключено. Зауважимо, що опис `Compile` можна доповнювати своїми правилами переписувань, що дає змогу програмісту вносити свої доповнення у перелік правил скорочень для препроцесування.

Процедури та функції. Процедури АПЛАН визначені наступним чином [25]:

```
<procedure definition>::= proc(<formal parameters list>)
                                  <local names> <statement>
<local names> ::= loc(<local names list>) | <empty>
<formal parameter> ::= <identifier>
```

Наприклад:

```
proc(a, b)loc(p)(
  P --> a,
  a --> b,
  b --> p
  );
```

Ім’я процедури зв’язується з її означенням оператором присвоєння

```
Name Swap;
Swap := proc(a, b)loc(p)(
   P --> a,
   a --> b,
   b --> p
 );
```

Існує два способи визначення функцій – або через системи правил переписування, або через процедури, які повертають значення. Значення повертає оператор `return`, який можна використовувати у трьох формах:

`return, return A, return(A),`

де `A` – ім’я або алгебраїчний вираз. Приклади функцій:

```
Name rsMax;
rsMax := rs(a,b)(a > b -> (a,b) = a,
      (a,b) = b
);

Name fnMax;
fnMax := proc(a, b)loc(p)
  a > b -> p := a
       else p := b,
  return p
 );
```

Відмітки та їх інтерпретатори. Як ми знаємо, програміст має змогу визначати власні операції шляхом об’яв відміток у розділі `Mark.` З кожною відміткою можна зв’язати її інтерпретатор за допомогою оператора процедури `markcan` (системний файл `proc_tbl.tbl`). Ця процедура має синтаксис

`markcan(`<відмітка>`,`<система інтерпретуючих правил переписувань>`);`

Наприклад:

```
MARK Sharp(2, 51, ()#());
rsSymSub = rs(a, b)(
    a > b -> a#b = a - b,
     a#b = b - a
);

markcan(()#(), rsSymSub);
```
/зв’язує операцію “#” з інтерпретатором `rsSymSub`/.
```
x := u#v; prn(x);
markcan(()#(),can0); //робить операцію “#” неінтерпретованою
x := u#v; prn(x);
markcan(()#(),sub_can);
```
/ зв’язує операцію “#” з інтерпретатором віднімання/
```
x := u#v; prn(x);
. . .
```

2.6. Система програмування АПС

Систему алгебраїчного програмування АПС розроблено у відділах 100,105 Інституту кібернетики НАН України під керівництвом акад. О.А. Летичевського. в 80-х роках XX сторіччя [19]. Наукові засади та опис системи наведено у [22 - 24]. На протязі багатьох років АПС використовували для прототипування різноманітних алгебраїчних алгоритмів у наукових дослідженнях [25 - 31]. Досвід використання АПС для цих цілей показав її ефективність як виробничої системи програмування ядра прикладних систем комп’ютерної алгебри.

Практика програмування алгебраїчних обчислень показала необхідність формалізації загальних підходів по реалізації цих задач, побудови математичної моделі алгебраїчних обчислень та методології її застосування. Розв'язанню цих проблем присвячено наступну главу.

2.7. Висновки

1. Використання систем правил переписування у якості основного засобу програмування символьних перетворень є важливою особливістю мови алгебраїчного програмування. Тому мови алгебраїчного програмування є основним інструментом реалізації прототипів програм комп’ютерної алгебри.

Глава 3. Математичні моделі та методи алгебраїчних обчислень

3.1. Багатосортні алгебри як математична модель алгебраїчних обчислень

Тісний зв'язок між абстрактною алгеброю, математичною логікою та програмуванням відзначено досить давно [36-37]. Зокрема, адекватною математичною моделлю алгебраїчних обчислень, основаних на символьних перетвореннях, виявилось поняття багатосортної (упорядковано-сортної) алгебраїчної системи. Ця модель є стандартною для теорії абстрактних типів даних [38-42] та мов логіко-алгебраїчних специфікацій [43-46]. Підхід до побудови алгебраїчних специфікацій алгебраїчних обчислень, що пропонується у цій книжці, відрізняється від стандартного у таких деталях.

По-перше, уточнюється поняття *розширення* багатосортних алгебр таким чином, що стає можливим формулювання у достатньо загальному вигляді алгоритма синтезу алгебраїчної програми – *інтерпретатора алгебраїчної операції*.

По-друге, розглядаються специфікації як абстрактних алгебр, визначених *аксіоматично*, так і конструктивних алгебр, визначених *конструкторами* елементів та *інтерпретаторами* операцій. Відзначимо, що вперше у явному вигляді ієрархію абстрактиних алгебраїчних систем як опис структури алгебраїчних типів було використано в системі комп'ютерної алгебри AXIOM [47, 48]. По суті це – ієрархія наслідування. Ми доповнюємо ієрархічно визначену в термінах наслідування аксіоматику алгебр конструктивними засобами [49-52]. Це дає змогу уточнити відомий у математиці конструктивний (генетичний) підхід до визначення алгебраїчних систем як алгоритм верифікації інтерпретаторів алгебраїчних операцій [52].

Нарешті, ми пропонуємо у явному вигляді визначати перетворення алгебраїчних типів як *ізоморфних* або *гомоморфних перетворень*.

3.1.1. Сигнатури багатосортних алгебр

Означення 3.1. Нехай $U = \{u_{1,\ldots,}u_k\}$ – скінченна множина символів, яка називається сигнатурою сортів. Символи $u_l, l \in \{1,\ldots,k\}$ називаються іменами сортів або просто сортами.

Далі будемо користуватися, зокрема, такими базовими сортами:

Variable – ім'я сорту-множини змінних,

Bool – ім'я сорту-множини логічних значень,

Nat – ім'я сорту-множини натуральних чисел,

Int – ім'я сорту-множини цілих чисел,

Real – ім'я сорту-множини дійсних чисел.

Інші імена сортів будемо вводити при означенні відповідних алгебраїчних понять.

Означення 3.2. Нехай, далі, $S = \{S_{u_1},...,S_{u_k}\}$ – скінченне сімейство множин, індексованих іменами сортів, які називаються областями значень відповідних сортів.

$S_{Variable}$ – множина змінних,
S_{Bool} – множина {False, True},
S_{Nat} – множина натуральних чисел,
S_{Int} – множина цілих чисел,
$S_{\mathrm{Re}al}$ – множина дійсних чисел.

Означення 3.3. (Частковою) багатосортною операцією f над сімейством S називається (часткове) відображення $f : S_{u_1} \times S_{u_2} \times ... \times S_{u_m} \to S_v$, де $u_1,...,u_m, v \in \mathbf{U}$ – сорти аргументів та області значень операції f, відповідно, а m – арність операції f.

Тип операції визначається переліком імен сортів її аргументів та іменем сорту області її значень. Тип операції f будемо позначати через $(u_1,...,u_m) \to v$. Сигнатурою Σ операцій називається скінченна множина символів операцій разом з відображенням, яке кожному символу $\varphi \in \Sigma$ співставляє багатосортну операцію f_φ разом з її типом (якщо φ символ операції, то вираз $\varphi : (u_1,...,u_m) \to v$ означає, що цьому символу співставлено операцію типу $(u_1,...,u_m) \to v$.

Багатосортною операцією, є, наприклад, операція множення у векторному просторі. Якщо *VectorSpace* – ім'я сорту-множини векторів над полем *Real* дійсних чисел, то операція множення *Mult* "*" задає відображення

$$Mult : \mathrm{Re}al \times VectorSpace \to VectorSpace$$

Надалі будемо користуватися більш звичними, тобто традиційними, математичними позначеннями операцій. Оскільки множення вектора на скаляр є бінарною операцією з інфіксною формою запису, маємо:

$$\mathrm{Re}al * VectorSpace \to VectorSpace.$$

Означення 3.4. Визначимо сорт *Bool* з областю значень $S_{Bool} = \{True, False\}$. Багатосортним предикатом P називається відображення $P : S_{u_1} \times ... \times S_{u_m} \to S_{Bool}$, де $u_1,...,u_m \in \mathrm{U}$, послідовність $u_1,...,u_m$ визначає тип предикату, а число його m – арність. Сигнатура Π багатосортних предикатів визначається аналогічно сигнатурі операцій як множина операцій символів предикатів, яким поставлені у відповідність багатосортні предикати разом з їх типами.

Означення 3.5. Багатосортною алгебраїчною системою (БАС) A називається четвірка $A=<S,U,\Sigma,\Pi>$, де S – множина сортів, індексованих символами множини U, $\Sigma=\{\varphi_1,...,\varphi_l\}$ – сигнатура багатосортних операцій, $\Pi=\{\pi_1,...,\pi_p\}$ – сигнатура багатосортних предикатів.

Примітка. Оскільки сорт *Bool* можна включити до множини сортів, предикати можна розглядати як багатосортні операції. Тому, об'єднавши сигнатури операцій та предикатів, разом з терміном БАС далі ми будемо вживати також термін «багатосортні алгебри».

Означення 3.6. Нехай $A=<S,U,\Sigma>$ багатосортна алгебра і $u,v\in U$ символи сортів цієї алгебри. Будемо казати, що сорт v залежить від сорту u, якщо одна з операцій сигнатури Σ має тип $u_1\times...\times u\times...\times u_m\to v$. Через U_v позначимо підмножину сортів, які залежать від v. Підмножину елементів Σ, що мають тип виду (1), позначимо через Σ_v, а сімейство областей значень сортів U_v позначимо через S_v. Обмеженням A_v багатосортної алгебри A на сорт v називається багатосортна алгебра $A_v=<S_v,U_v,\Sigma_v>$.

Таким чином, багатосортна алгебра A може бути представлена набором обмежень (алгебр) $A_v, v\in U$, тобто $A=<A_{u_1},...,A_{u_k}>$.

Приклад 3.1. Припустимо, що треба побудувати програмну систему, яка підтримує спрощення цілих алгебраїчних та тригонометричних виразів. Ядро цієї системи має підтримувати обчислення у кільці поліномів та кільці тригонометричних виразів багатьох змінних над полем раціональних чисел. Отже специфікаціям підлягають такі БАС – обмеження на вказані сорти:

1. *MultiPolynom* – кільце поліномів багатьох змінних. Сигнатура операцій:

MultiPolynom + MultiPolynom $\to$ *MultiPolynom*
MultiPolynom - MultiPolynom $\to$ *MultiPolynom*
*MultiPolynom * MultiPolynom* $\to$ *MultiPolynom*
MultiPolynom ^ Int $\to$ *MultiPolynom*
*Rat * MultiPolynom* $\to$ *MultiPolynom*
MultiPolynom / Rat $\to$ *MultiPolynom*

2. *MultiTrig* – кільце тригонометричних поліномів багатьох змінних. Сигнатура операцій:

Sin(LinComb) $\to$ *MultiTrig*
Cos(LinComb) $\to$ *MultiTrig*
MultiTrig + MultiTrig $\to$ *MultiTrig*
MultiTrig - MultiTrig $\to$ *MultiTrig*
*MultiTrig * MultiTrig* $\to$ *MultiTrig*
MultiTrig ^ Int $\to$ *MultiTrig*
*Rat * MultiTrig* $\to$ *MultiTrig*
MultiTrig / Rat $\to$ *MultiTrig*

3. *LinComb* – векторний простір лінійних комбінацій багатьох змінних (аргументи тригонометричних поліномів). Сигнатура операцій:

Pi

LinComb + LinComb $\rightarrow$ *LinComb*

LinComb - LinComb $\rightarrow$ *LinComb*

*Rat * LinComb* $\rightarrow$ *LinComb*

LinComb / Rat $\rightarrow$ *LinComb*

4. *Rat* – поле раціональних чисел (коефіцієнти поліномів та тригонометричних поліномів). Сигнатура:

Rat + Rat $\rightarrow$ *Rat*

Rat - Rat $\rightarrow$ *Rat*

*Rat * Rat* $\rightarrow$ *Rat*

Rat ^ Int $\rightarrow$ *Rat*

Rat / Rat $\rightarrow$ *Rat* *//Знаменник не дорівнює нулю*

5. *RatBool* – доповнення Bool предикатами рівності та порядку на *Rat*:

Rat = Rat $\rightarrow$ *Bool*

Rat < Rat $\rightarrow$ *Bool*

Rat > Rat $\rightarrow$ *Bool*

Rat <= Rat $\rightarrow$ *Bool*

Rat => Rat $\rightarrow$ *Bool*

Відношення залежності сортів породжує структуру залежності на множині алгебр $A_u, u \in U$: алгебра A_v залежить від алгебри A_u якщо сорт v залежить від сорту u. Якщо відношення залежності не має циклів, багатосортну алгебру можна будувати крок за кроком (інкрементно), будуючи алгебру A_v, якщо алгебри, від яких A_u залежить, уже побудовані.

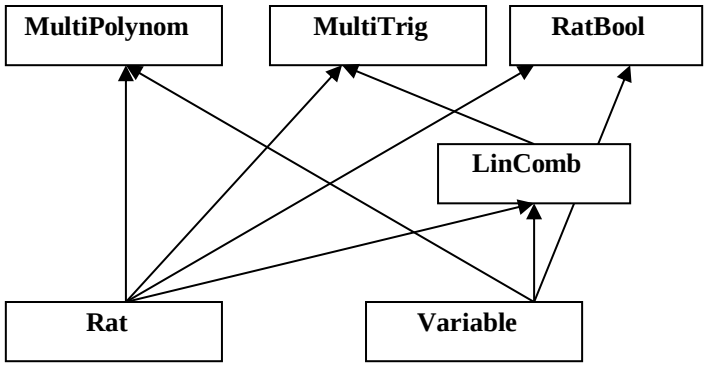


Рис 3.1 Діаграма відношення залежності алгебр прикладу 3.1.

3.1.2. Аксіоми та конструкції багатосортних алгебр

Для побудови алгебр A_u використовуємо їх аксіоматичні та конструктивні описи (означення).

Означення 3.7. *Аксіомою* алгебри A_u називається тотожність або умовна тотожність у сигнатурі Σ_u. *Аксіоматичним описом* алгебри A_u, за означенням, є скінченний набір аксіом (система аксіом) алгебри A_u.

Ми будемо користуватися алгебраїчною термінологією та відповідними системами аксіом з підручників [4 - 8].

Для визначення конкретної алгебри разом з аксіомами часто використовують так звані постулати мінімальності. Наприклад, поле раціональних чисел можна визначити як мінімальне поле, яке містить множину натуральних чисел.

Конструктивний опис алгебри A_u – це означення конструктора сорту S_u (тобто визначення термів, множина яких утворює сорт S_u) та множини інтерпретаторів операцій Σ_u.

Означення 3.8. Сигнатурою конструкторів T_u називається скінченна множина символів операцій разом з відображенням, яке кожному символу $\tau \in T_u$ співставляє символ сорту u разом з переліком символів сортів його аргументів (якщо τ – символ операції, то вираз $u = \tau(u_1,...,u_m)$ означає, що цьому символу співставлено символ сорту u та символи сортів його аргументів $u_1,...,u_m$.

Конструктором сорту S_u алгебри A_u називається система рівностей**,** яка визначає синтаксично елементи сорту S_u як терми у сигнатурі T_u. Отже, сорт S_u – це множина термів у (власній) сигнатурі T_u конструктора сорту S_u.

Означення 3.8 є ключовим у нашому підході до специфікації алгебраїчних обчислень. Тому ми надамо необхідні приклади та пояснення.

Приклад 3.2. Поле *Rat* раціональних чисел. Елементи цього поля – раціональні числа у вигляді звичайних дробів. Конструктор сорту визначає стандартну форму представлення елементу цього сорту. Частіше за все, це – *канонічна форма*. Таким чином,

$$S_{Rat} = \{\frac{p}{q} : p \in S_{Int}, q \in S_{Nat}, GCD(p,q) = 1\}$$

Роль конструктора сорту грає символ «-» (горизонтальна риска). Цей же символ математики використовують для позначення операції ділення, зокрема, у *Rat.* Це явище у математиці є розповсюдженим. Таку ж роль грає, наприклад, символ кореня, символи додавання та множення ($a + b * \sqrt{2}$). І ці приклади можна продовжувати. Виявилося, однак, що для задачі специфікацій алгебраїчних обчислень це не зовсім зручно. Конструктори сортів мають бути позначеними окремо. Тому вводимо окремо поняття сигнатури операцій Σ та сигнатури конструкторів T. Зокрема, для конструктора сорту *Rat* ми використовуємо *подвійну нахильну риску*:

$$S_{Rat} = \{p // q : p \in S_{Int}, q \in S_{Nat}, GCD(p,q) = 1\}$$

Ще однією важливою обставиною є те, що у стандартних формах представлення елементів сортів синтаксичні аспекти означення, які визначаються символами конструкторів, практично завжди поєднуються з семантичними аспектами, що задаються у вигляді контекстних умов, тобто предикатів. У нашому прикладі таким предикатом є рівність $GCD(p,q) = 1$.

Приклад 3.3. Кільце *Polynom* поліномів однієї змінної над полем *Rat*. Елементи цього поля – поліноми, які представлені звичайно у вигляді суми мономів, упорядкованих у порядку спадання степенів.

$$S_{Polynom} = \{M_0 + M_1 + ... + M_k : M_i \in S_{Monom}, \deg(M_i) > \deg(M_{i+1})\}$$

У специфікаціях сорту *Polynom* це означення має бути рекурсивним, оскільки треба видалити з формули символ “три крапки” та визначити окремо випадок, коли поліном містить лише один моном. Як ми побачимо, при такому підході окремо треба визначити ще поняття степеня полінома.

$$S_{Polynom} = \{Q : Q = M + +P, M \in S_{Monom}, P \in S_{Polynom}, \deg Q \overset{df}{=} \deg M;$$

$$\deg(M) > \deg(P)\} \cup S_{Monom}$$

Отже, для визначення носіїв сортів будемо користуватися спеціальною мовою специфікацій, яка допускає нерекурсивні та рекурсивні синтаксичні визначення елементів сортів, визначення функцій доступу (зокрема, для формулювання контекстних умов) та контекстні умови. Наведемо визначення сортів наших прикладів цією мовою:

```
Rat r ={
   (Int a)//(Nat b);          // Конструктор сорту
   Num(r) = a, Den(r) = b;          // Функції доступу
   GCD(a, b) = 1              // Контекстна умова
};
Monom M =  {
  (Rat c)$(Const Variable x)^^(Nat n); // Конструктор сорту
   Coef(M) = c,          // Функції доступу
   Var(M) = x,
   Deg(M) = n
};
Polynom P = {
 Monom M)++(Polynom Q)|(Monom M);  // Конструктор сорту
 LeadMon(P) = M,                   // Функції доступу
 LeadCoef(P) = Coef(M),
 Deg(P) = Deg(M);
 Deg(P) > Deg(Q)                   // Контекстна умова
};
```

Для того, щоб реалізувати обчислення в деякій алгебрі $A_v, v \in U$, потрібно реалізувати алгоритми виконання кожної її операції таким чином, щоб виконувались аксіоми цієї алгебри.

Означення 3.9. Інтерпретатором операції сигнатури Σ_u називається функція, яка реалізує алгоритм виконання цієї операції.

Інтерпретатори операцій визначають засобами мови *APLAN* (п.2.5). Отже, ми включаємо цю мову у мову специфікацій.

Для аксіоматичного та конструктивного визначення алгебри A_v до її означення ми включаємо скінченні множини аксіом Ax_v та інтерпретаторів I_v. Отже, багатосортна алгебра A_v визначається наступним чином: $A_v = < S_v, U_v, T_v, \Sigma_v, Ax_v, I_v >$.

3.2. Методи побудови багатосортних алгебр

Побудова структури БАС полягає у специфікуванні, прототипуванні та реалізації алгебраїчних обчислень. Специфікування структури багатосортних алгебр здійснюється у термінах *розширень, гомоморфізмів, ізоморфізмів* та *наслідування* багатосортних алгебр [49-52, 57-59]. Таким чином, разом з діаграмами залежності, які є графічними моделями специфікацій сигнатур операцій та конструкторів, будуються діаграми розширень, діаграми морфізмів (тобто ізоморфізмів та гомоморфізмів) та діаграми наслідування. Ці діаграми є графічними моделями специфікацій конструкторів алгебр та інтерпретаторів відповідних алгебраїчних операцій.

3.2.1. Метод розширення БАС

Фундаментальну роль ієрархії розширень БАС відзначено в [49]. Наш підхід полягає у конструктивному уточненні поняття розширення як *конструкції* та *вкладення*.

Означення 3.10. Нехай A_u та A_v – багатосортні алгебри. Багатосортна алгебра A_v називається *конструктивним розширенням* багатосортної алгебри A_u, якщо $S_u \subseteq S_v$ та для будь-якої пари операцій f_1 та f_2 типів відповідно $f_1 : (u_1,...,u_m) \to u$ та $f_2 : (v_1,...,v_m) \to v$, якщо $S_{u_1} \subseteq S_{v_1},...,S_{u_m} \subseteq S_{v_m}$, то $\forall (a_1,...,a_m) \in S_{u_1} \times ... \times S_{u_m}$ має місце рівність $f_1(a_1,...,a_m) = f_2(a_1,...,a_m)$.

Вкладенням називається ізоморфне відображення $\mathrm{Re}\, d : S_u \to S'_v$, яке відображає S_u на підмножину $S'_v \subset S_v$. Обмеження алгебри A_v на підмножину $E_1(x),...,E_k(x)$, ізоморфне A_u, визначається системою умовних тотожностей $E_1(x),...,E_k(x)$: $S'_v = \{a \in S_v \mid E_1(a),...,E_k(a)\}$. Застосування системи

$E_1(x),...,E_k(x)$ як системи переписувань «спрощує» терм $a \in S'_v$ до терма $a' \in S_u$: $\mathrm{Re}\,d^{-1}(a) = a'$.

Конструктивне визначення розширення A_v полягає у визначенні конструктора A_v та вкладення A_u в алгебру A_v. На рис. 3.2 подвійною стрілкою позначено той факт, що алгебра A_v є розширенням алгебри A_u, в алгебрі A_v визначено конструкцію $v = \tau(u_1,...,u,...,u_m)$ та вкладення $Red_{u,v}$.

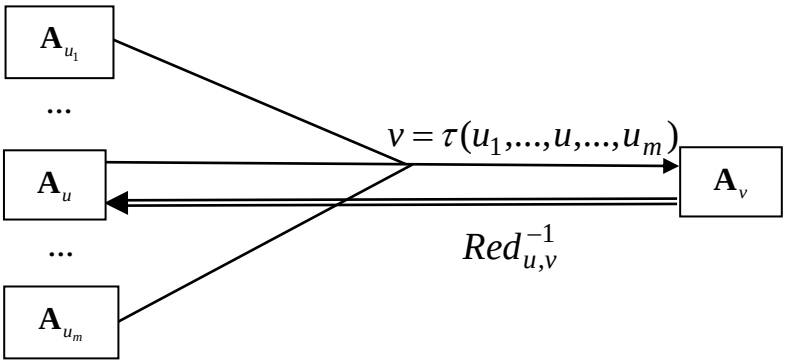


Рис 3.2. Фрагмент діаграми конструктивних розширень

Приклад 3.4. Розглянемо конструктор поля *Rat* (приклад 3.2). У відповідності до означення він визначає конструкцію *Rat,* аргументами якої є сорти *Int* та *Nat*. Доповнимо специфікації сорту *Rat* вкладенням $\mathrm{Re}\,d : Rat \to Int$, яке визначено рівністю $\mathrm{Re}\,d(a//1) = a$. Таким чином сорт *Rat* визначено як конструктивне розширення сорту *Int.*

Примітка. Уточнення «*коструктивне*», якщо з контексту очевидно, що мова йде про *конструктиве розширення*, надалі будемо опускати.

Розглянемо тепер конструктор кільця *Polynom* (приклад 3.3). Він визначає рекурсивно конструкцію *Polynom*, аргументом якої є сорт *Monom.* Доповнимо специфікації сорту *Polynom* вкладенням $\mathrm{Re}\,d : Polynom \to Monom$, яке визначено рівністю $\mathrm{Re}\,d(M++0) = M$. Отже, сорт *Polynom* визначено як розширення сорту *Monom.* У свою чергу, сорт *Monom* є розширенням сорту *Degree* з функцією *Red*, визначеною рівністю 1$x^^k = x^^k, розширенням сорту LinMonom з функцією *Red*, визначеною рівністю *a$x^^1 = a$x* та розширенням сорту *Rat* з функцією *Red*, визначеною рівністю a$x^^0 = a. Сорти *Degree* та *LinMonom* є розширенням сорту *Variable* з функціями редукції, заданими відповідно рівностями *x^^1 = x* та *1$x = x*. Отже, діаграма розширень має вид, наведений на рис. 3.3.

Механізм *конструктивних розширень* є одним з основних методів специфікацій багатосортних алгебр. Зокрема, він дозволяє визначити перевантажені алгебраїчні операції та функції приведення алгебраїчних типів.

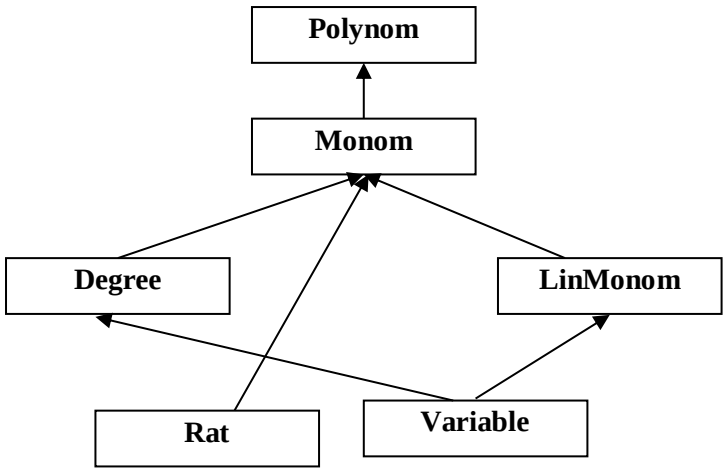


Рис 3.3. Діаграма конструктивних розширень прикладу 3.4.

3.2.2. Метод морфізмів БАС

Означення 3.11. Нехай A_u та A_v – багатосортні алгебри. A_v називається *гомоморфною* A_u, якщо існує таке відображення $H: S_u \to S_v$, що для будь-якої операції f_1 типу $\varphi: (u_1,...,u,...,u,...,u_m) \to u_{m+1}$ існує операція f_2 типу $\varphi: (u_1,...,v,...,v,...,u_m) \to u_{m+1}$ та виконується рівність $H(f_1(a_1,...,a,...,b,...,a_m)) = f_2(a_1,...,H(a),...,H(b),...,a_m)$, де

$$(a_1,...,a,H(a),b,H(b),...,a_m) \in S_{u_1} \times ... \times S_u \times S_v \times S_u \times S_v \times ... \times S_{u_m}.$$

Багатосортні алгебри A_u та A_v називаються ізоморфними, якщо існує взаємно однозначний гомоморфізм: $M: S_u \leftrightarrow S_v$.

Гомоморфізми та ізоморфізми багатосортних алгебр – зручний засіб для опису алгебраїчних обчислень в багатьох ситуаціях. З погляду програмування, це – функції перетворення типів.

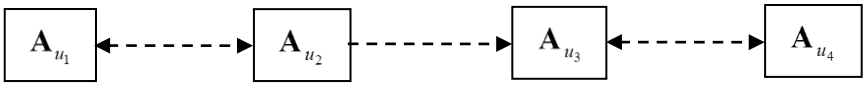


Рис. 3.4. Фрагмент діаграми морфізмів

Двоспрямована пунктирна стрілка на рис. 3.4 означає той факт, що між алгебрами A_{u_1} та A_{u_2} визначено ізоморфізм, а односпрямована пунктирна стрілка означає, що визначено гомоморфізм з A_{u_2} у A_{u_3}.

Приклад 3.5. Кільце *Polynom* поліномів однієї змінної над полем *Rat*. Елементи цього поля визначені конструкторами, наведеними у прикладі 3.3. Зокрема, конструктор сорту *Monom* визначає моном виду $a*x^k$. У такому вигляді звичайно здійснюють введення поліномів-даних і отримують поліноми-результати обчислень. Відомо, однак, що більш ефективним представленням поліномів однієї змінної у пам'яті є так зване розріджене представлення, у якому мономи є парами (коефіцієнт, степінь), а поліном –

упорядкованою послідовністю мономів. Таким чином, маємо справу з двома ізоморфними представленнями, по суті, однієї алгебри і проблема полягає у побудові специфікацій, які адекватно відображають такий метод обчислень: треба спочатку здійснити перетворення введених даних у внутрішнє (ефективне) представлення, здійснити обчислення, а потім здійснити обернене перетворення у зовнішнє представлення. Специфікації функцій перетворень H та H^{-1} задаються рівностями:

$$H(a*x^k)=((a,k),x), \quad H^{-1}((a,k),x)=a*x^k$$

$$H((m,x)++P)=((m,\ H(P)),x) \text{ де } M=(m,x), m=(a,k)$$

Іншим прикладом використання ізоморфізмів є ізоморфізми, за допомогою яких визначаються три представлення раціональних чисел, якими оперує шкільний курс алгебри: звичайні дроби, змішані дроби та десяткові періодичні дроби.

3.2.3. Метод наслідування БАС

Означення 3.12. Будемо казати, що алгебра $A_v =< S_v,U_v,T_v,\Sigma_v,Ax_v,I_v >$ наслідує алгебру $A_u =< S_u,U_u,T_u,\Sigma_u,Ax_u,I_u >$ якщо виконується хоча б одна з умов:

1) $\Sigma_u \subseteq \Sigma_v$;
2) $Ax_u \subseteq Ax_v$;
3) якщо T_v належить тип означення $v=\tau(u_1,...,v',...,u_n)$, T_u належить тип означення $u=\tau(u_1,...,u',...,u_n)$ та алгебра A_v наслідує алгебру $A_{u'}$;
4) якщо T_u належить тип означення $u=\tau(u_1,...,u',...,u_n)$, $u'=\tau_1(v_1,...,v_m)$ та $v=\tau(u_1,...,\tau_1(v_1,...,v_m),...,u_n)$.

Наслідування є одним з основних методів об'єктно-орієнтовного програмування. Зауважимо, що БАС може наслідувати декілька інших БАС (множинне наслідування).

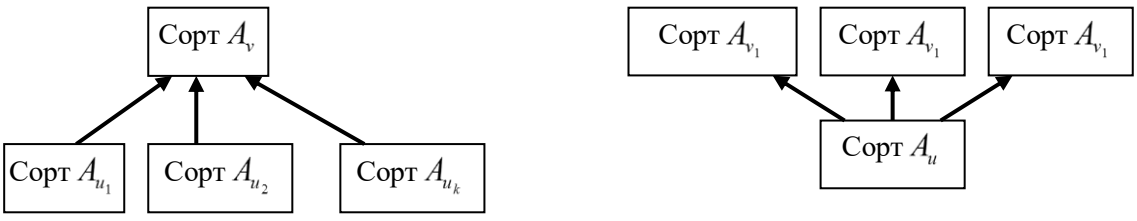


Рис 3.5. Фрагмент структури простого та множинного наслідування

Товсті одинарні стрілки, проведені з алгебр $A_{u_1},...,A_{u_k}$ до алгебри A_v на рис. 3.5, означають, що алгебри $A_{u_1},...,A_{u_k}$ наслідують алгебру A_v. Такі ж стрілки, проведені з алгебри A_u до алгебр $A_{v_1},...,A_{v_k}$ на рис. 3.5, показують множинне наслідування: алгебра A_u наслідує алгебри $A_{u_1},...,A_{u_k}$.

Наслідування використовується, зокрема, коли треба доповнити систему аксіом новою аксіомою, по-друге, коли треба доповнити сигнатуру операцій

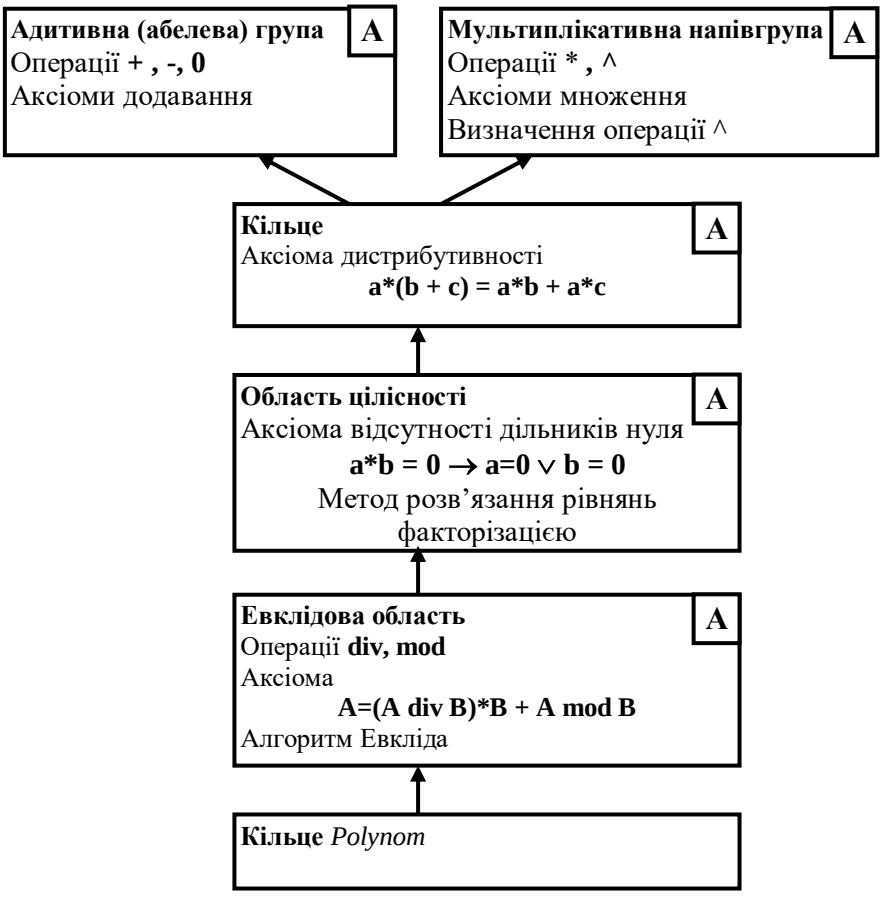
новою операцією. На першому етапі специфікування будують необхідну ієрархію абстрактних алгебр, яка визначає, насамперед, сигнатури та аксіоми алгебр. Крім того, абстрактні алгебри визначають вид та властивості виділених констант, а також ті важливі алгоритми, які не потребують деталей представлення даних.

*Приклад 3.6***.** Розглянемо аксіоматичне визначення кільця з п.1.2. як групи за операцією «+» та напівгрупи за операцією «*» з додатковою аксіомою дистрибутивності. У термінах наслідування кільце визначається як нащадок адитивної групи та мультиплікативної підгрупи (множинне наслідування). У свою чергу, кільце породжує область цілісності, а область цілісності – евклідову область.

Побудова ієрархії абстракцій (рис 3.6) дозволяє визначити сигнатуру, аксіоми *Polynom,* побудувати повторно використовувані інтерпретатори обчислень з нулем та одиницею, алгоритми Евкліда, розв'язування рівнянь методом факторизації тощо.

Наслідування також використовується на етапі реалізації програмного модуля виділенням абстракцій та їх ефективною реалізацією (наприклад, на C++). Більш детально ми опишемо цей метод нижче.

Отже, специфікації полягають у побудові структури багатосортних алгебр у термінах розширень, морфізмів та наслідування багатосортних алгебр, побудові прототипу мовою APLAN в системі алгебраїчного програмування



APS та його ефективної реалізації.

Рис. 3.6. Ієрархія наслідування абстрактних алгебр,
яка специфікує сорт *Polynom*.

3.3. Вивід інтерпретаторів алгебраїчних операцій у конструктивних розширеннях алгебр

3.3.1. Статичні та динамічні конструктивні розширення

Означення 3.13. Розширення B алгебри A називається статичним (нерекурсивним), якщо у його конструкторі $B = \tau(A_1,...,A,...,A_n)$ жоден з аргументів не співпадає з B.

Приклад 3.7. Статичні розширення.

1. Поле *Rat* є статичним розширенням кільця *Int*, оскільки $Rat\,R = (Int\,A)//(Nat\,B)$

2. Напівгрупа мономів *Monom* однієї твірної є статичним розширенням поля коефіцієнтів *Coef*, оскільки $Monom\,M = (Coef\ a)\$(Var\ x)\hat{}(Nat\ N)$

Означення 3.14. Розширення B алгебри A називається динамічним (рекурсивним), якщо у його конструкторі $B = \varphi(A_1,...,A,...,A_n)$ щонайменше один з аргументів співпадає з B.

Конструктор динамічних розширень є рекурсивним визначенням, отже, місить як базову, так і рекурентну частини.

Приклад 3.8. Динамічні розширення.

3. Векторний простір *LinComb* лінійних комбінацій багатьох змінних над полем *Coef* є лінійним динамічним розширенням одновимірного лінійного простору *LinMonom*, елемент якого має вигляд $a\$x$. Елемент $w \in LinComb$ має вид $w = a_1\$x_1 + +a_2\$x_2 + +... + +a_m\$x_m$. Таким чином, його конструктор є рекурсивним означенням:

$$LinComb\ w = (Coef\ a)\$(Variable\ x) + +LinComb\ w|(Coef\ a)\$(Variable\ x)$$

4. Кільце многочленів однієї змінної *Polynom* над полем *Coef*. Це кільце в алгебрі позначається через *F[x]* .

$$Polynom\ w = (Monom M) + +(Polynom\ w) \mid (Monom M)$$

Практика використання динамічних розширень показала корисність їх подальшої класифікації як лінійних та бінарних (динамічних) розширень.

Означення 3.15. Динамічне розширення B БАС A називається *лінійним*, якщо у його конструкторі $B = \varphi(A_1,...,A,...,A_n)$ у точності один з аргументів співпадає з B.

Динамічне розширення B алгебри A називається *бінарним*, якщо у його конструкторі $B = \varphi(A_1,...,A,...,A_n)$ у точності два аргументи співпадають з B.

Приклади 3, 4, наведені при його означенні, є прикладами лінійних динамічних розширень. Розглянемо приклад бінарного динамічного розширення:

5. Числове поле *Rad*, елементами якого є лінійні комбінації квадратних коренів натуральних чисел, вільних від квадратів, з раціональними коефіцієнтами, можна представити як бінарне розширення поля *Rat* за допомогою наступної конструкції. Нехай $p_1, p_2, ..., p_n, ...$ – послідовність усіх простих чисел, розташованих у порядку зростання. Позначимо також через Q поле раціональних чисел. Введемо наступні позначення: $Rad_0 = Q,\ Rad_n = \{r : r = a + b * \sqrt{p_n},\ a, b \in Rad_{n-1}, n = 1,2,...\}$. Поле *Rad* є нескінченним об'єднанням зростаючої послідовності полів Rad_n.

$$Rad = \bigcup_{n=0}^{\infty} Rad_n,\ Rad_0 \subset Rad_1 \subset ... \subset Rad_n \subset ... \qquad (3.1)$$

Отже, конструктор *Rad* має вид

$$Rad\ r = (Rad\ a) + (Rad\ b) * \sqrt{Nat\ p}\ |(Rat\ q)$$

Зауважимо, що послідовність розширень (3.1) є послідовністю скінченних алгебраїчних розширень полів коренями поліномів $x^2 - p_n = 0$.

Формули (3.1) безпосередньо узагальнюються на довільні динамічні розширення. Якщо B є динамічним розширенням A *з конструктором* $B = \varphi(A_1, ..., B, ..., A_n)$, зростаючу послідовність $B_0 \subset B_1 \subset ... B_n \subset ...$ визначимо таким чином:

1. $B_{(0)} = A,$ (3.2)
2. $B_{(n+1)} = \varphi(A_1, ..., B_{(n)}, ..., A_n)$ (3.3)

Вкладення $\mathrm{Re}d: A \to B$ визначає вкладення $\mathrm{Re}\,d_i : B_{i+1} \to B_i$, звідки безпосередньо випливає представлення A у вигляді об'єднання зростаючої послідовності алгебр, кожна з яких є статичним розширенням попередньої.

$$B = \bigcup_{n=0}^{\infty} B_n,\ B_0 \subset B_1 \subset ... \subset B_n \subset ... \qquad (3.4).$$

Таким чином, динамічні розширення алгебр є послідовностями статичних розширень. Цей факт дозволяє, як побачимо далі, використовувати загальну схему реалізації динамічних розширень (3.2)-(3.4), виводячи відповідні системи правил переписувань.

3.3.2. Вивід інтерпретаторів операцій в статичних розширеннях

Приклад 3.9. Специфікації сорту *Rat* та вивід обчислень з раціональними числами. Специфікації сорту *Rat* визначають цей сорт як поле, лінійний порядок та статичне розширення *Int*.

```
Sort Rat:: Field, LinOrd; //Наслідування
Constructor{
    Rat r = (Int a)//(Nat b);              // Конструктор сорту
    a//1 = a;                              // Функція вкладення RatToInt
    Num(r) = a, Den(r) = b;                // Функції доступу
    GCD(a, b) = 1                          // Контекстна умова
    Form: Num(Form(r))∈ Int, Den(Form(r))∈ Nat,
    GCD(Num(Form(r), Den(Form(r)) = 1;
    };
    Operations
    Add:  a//b + c//d = Form((a*d + b*c)//(b*d));
    Sub:  a//b - c//d = Form((a*d - b*c)//(b*d));
    Mult: a//b * c//d = Form((a*c)//(b*d));
    Div:  a//b / c//d = Form((a*d)//(b*c));
    Div:  a/b = Form(a//b), a/0 = Exeption('Divison by zero');
    Pow:  n >= 0 -> (a//b)^n  = (a^n//b^n),
          n < 0 ->   (a//b)^n  = (b^-n//a^-n);
    Predicates
    Equ:  a//b == c//d = (a == c)&(b == d);
    Gre:  a//b > c//d = (a*d > b*c);
    Les:  a//b < c//d = (a*d < b*c);
    UnLes: a//b >= c//d = (a//b > c//d)∨( a//b == c//d);
    UnMor: a//b <= c//d = (a//b < c//d)∨( a//b == c//d);
```

Розглянемо вивід алгебраїчної програми операції додавання Add. Зі специфікацій маємо:

$$a//b + c//d = Form((a*d + b*c)//(b*d)); \quad (3.5)$$

$$a//1 = a. \quad (3.6)$$

Підставляючи ліву частину (3.6) замість першого операнду у (3.5), враховуючи, що $b = 1$, отримуємо:

$$a + c//d = Form((a*d + 1*c)//(1*d)).$$

Застосовуючи тотожність сорту *Int* $a*1 = 1*a = a$, отримуємо:

$$a + c//d = Form((a*d + c)//d).$$

Аналогічно для другого операнду:

$$a//b + c = Form((a + b*c)//b.$$

Виведені співвідношення є частинними випадками загального співвідношення (3.5) – специфікації операції додавання Add – результатом вкладення *RatToInt*. Разом з загальним співвідношенням вони визначають правила виконання додавання дробів:

```
Add:=rs{
        a//b + c//d = Form((a*d + b*c),(b*d)),
           a + c//d = Form((a*d + c),d),
           a//b + c = Form((a + b*c),b)
};
```

Наступні перетворення, що стосуються функції *Form*, є оптимізуючими. Покажемо, що виклики Form у другому та третьому правилах можна видалити. Для цього треба показати, що

$$a*d+c\in Int, d\in Nat, GCD(a*d+c,d)=1,$$

якщо

$$a\in Int, b\in Nat, GCD(a,b)=1, b=1, \quad c\in Int,\ d\in Nat, GCD(c,d)=1.$$

Спростимо умови 1-го операнду, використовуючи умову вкладення $b=1$. Отримаємо умову $a\in Int$. Отже, треба довести

$$a\in Int, c\in Int, d\in Nat, GCD(c,d)=1\rightarrow$$
$$a*d+c\in Int, d\in Nat, GCD(a*d+c,d)=1$$

Тобто

$$a\in Int, c\in Int, d\in Nat, GCD(c,d)=1\Rightarrow a*d+c\in Int,$$
$$a\in Int, c\in Int, d\in Nat, GCD(c,d)=1\Rightarrow d\in Int,$$
$$a\in Int, c\in Int, d\in Nat, GCD(c,d)=1\Rightarrow GCD(a*d+c,d)=1.$$

Легко побачити, що перші два умовиводи є очевидними, а третій можна скоротити:

$$GCD(c,d)=1\Rightarrow GCD(a*d+c,d)=1.$$

Однак, це є наслідком властивості функції *GCD*: $GCD(a*x+c,x)=GCD(c,x)$. Враховуючи доведені спрощення, отримаємо оптимізовані правила додавання дробів – інтерпретатор операції додавання в *Rat*:

```
Add:=rs{
       a//b + c//d = Form((a*d + b*c),(b*d)),
          a + c//d = (a*d + c)//d,
          a//b + c = (a + b*c)//b
};
```

Цілком аналогічно виводяться інтерпретатори усіх інших операцій сорту *Rat,* за виключенням операції ділення раціональних чисел. Ця операція відсутня у сигнатурі сорту *Int*. Тому її треба визначити як багатосортну. Крім того, треба специфікувати виключну ситуацію – ділення на нуль (див специфікації сорту *Rat*).

Звернемо тепер увагу на функцію *Form*. Визначення цієї функції зв'язує її з символом конструктора сорту. Роль цієї функції по суті полягає у канонізації елементу сорту. При виконанні правила загального виду ця функція викликається на результат операції. Тому функція *Form* є інтерпретатором символу конструктора сорту. Для зручності користування нею при позначенні результатів бінарних операцій з інфіксною формою запису треба ввести відповідний символ алгебраїчної операції. Зокрема, для символу конструктора сорту *Rat* «//» ми будемо користуватися позначенням «!/». Загальне правило додавання прийме вигляд $a//b+c//d=a*d+b*c!/(b*d)$. Знак " ! " завжди буде включатися до складу інфіксних позначень конструкторів сортів і завжди буде означати виклик інтерпретатора конструктора сорту. Таким чином, у системі правил

```
Add:=rs{
       a//b + c//d = (a*d + b*c)!/(b*d),
          a + c//d = (a*d + c)//d,
          a//b + c = (a + b*c)//b
};
```

інтерпретатор конструктора сорту викликається на результат виконання операції тільки у першому правилі.

Загальний алгоритм. Узагальнимо метод виводу алгебраїчної програми – інтерпретатора даної операції φ сорту v з його специфікацій за умови, що алгебру A_v визначено як статичне розширення алгебри A_u сорту u: $A_u \subset A_v$.

Специфікації сорту v визначають:

1. *Конструктор сорту* $v=\tau(u,w_1,...,w_k)$. Обов'язковим аргументом τ є сорт u. Для скорочення запису набор інших аргументів позначимо через $W=(w_1,...,w_k)$. Отже, $v=\tau(u,W)$. *Контекстну умову* конструктора позначимо через $\pi(u,W)$. *Функції доступу* до аргументів τ позначимо через arg(τ, i), i = 1, 2. $\arg(\tau,1)=u$, $\arg(\tau,2)=W$.

2. *Умова вкладення* $\rho(u,W)$ визначає вкладення $A_u \subset A_v$ умовною рівністю $\rho(u,W) \to \tau(u,W)=u$.

3. *Інтерпретатор конструктора сорту* $Form(x,p_1,...,p_k)=y$ формує елемент y сорту v з його аргументів $x,p_1,...,p_k$ таким чином, що виконується контекстна умова сорту v $\pi(x,p_1,...,p_k)$. Зауважимо, що функція *Form* має бути специфікована заданням конгруенції Θ на множині $S_u \times S_{w_1} \times ... \times S_{w_k}$: $(x,p_1,...,p_k) \overset{\Theta}{\approx} (x',p_1',...,p_k')$.

4. *Операція,* яка підлягає специфікації – $\varphi(x_1, ..., x_m, p_1,.., p_l) = y$. Ліва частина рівності визначає символ операції та її аргументи, права – результат операції. Ті аргументи, які належать сорту v, позначимо через $x_1, ..., x_m$. Інші аргументи позначимо через $P = (p_1,.., p_l)$. Вони грають роль параметрів. Нагадаємо, що змінна *y* також належить сорту v.

5. *Специфікація операції* φ задана системою рівностей

$$\begin{aligned}
&\varphi(x_1, ..., x_m, P) = y,\\
&x_1 = \tau(a_1, B_1),\\
&...\\
&x_m = \tau(a_m, B_m),\\
&t = f(a_1,..., a_m, B_1,..., B_m, P),\\
&R = G(a_1,..., a_m, B_1,..., B_m, P),\\
&...\\
&y = Form(t, R).
\end{aligned} \tag{3.7}$$

Результатом виводу є *алгебраїчна програма,* тобто система правил переписування, яка визначає значення y операції $\varphi(x_1, ..., x_m, p_1,.., p_l)$ за значеннями аргументів $x_1, ..., x_m, p_1,.., p_l$.

Власне алгебраїчна програма є системою з 2^m рівностей, кожна з яких є частинним випадком рівності $\varphi(\tau(a_1, B_1),..., \tau(a_m, B_m), P) = Form(t, R)$, які отримуються вкладеннями довільної підмножини $X_J = \{x_{j_1},..., x_{j_l}\}$ множини $X = \{x_1,..., x_m\}$ аргументів сорту $\boldsymbol{v}$ цієї операції у сорт u: $\varphi(x_1, ..., x_m, P) = Form(t, R)$, де

$$x_i = \begin{cases} x_i = a_i, i \in J \\ x_i = \tau(a_i, B_i),\ i \in [1..m] - J \end{cases},$$

$$C_i = \begin{cases} B_i^{(a)} : \rho(a_i, B_i^{(a)}) = true,\ i \in J \\ B_i \end{cases}. \tag{3.8}$$

Зауважимо, що умова вкладення $\rho(u, W)$ для кожного значення a сорту u визначає єдине значення B_a таке, що $\rho(a, B^{(a)}) = true$. У багатьох випадках цю умову формулюють як функцію. Отже, в результаті застосування алгоритму перебору усіх підмножин $X_j \subseteq X$ отримуємо систему правил $L_{X_J} = R_{X_J}, X_J \subseteq X$, яка визначає інтерпретатор операції φ арності m.

Зауважимо також, що у випадках, коли операція φ задається декількома умовними рівностями (кусково-визначені операції) або умовною рівністю (частково визначені операції), принципово нічого не змінюється. Алгебраїчна програма – інтерпретатор операції φ визначається перебором усіх варіантів вкладення аргументів цієї операції у сорт *v*. Прикладом кусково-визначеної

операції є специфікація операції *Pow*, а прикладом частково-визначеної операції – операція ділення *Div*.

```
a/0 = Exeption('Division by zero')
```

У загальному випадку інтерпретатор операції φ, визначений системою правил – умовних або безумовних рівностей $U_{X_J} \to L_{X_J} = R_{X_J}$, кожна з яких є частинним випадком специфікації операції.

Якщо операція φ визначена на базовому сорті u, останнє з правил інтерпретатора, яке відповідає випадку $X_J = X$, є вже інтерпретованим, тому це правило з інтерпретатора операції треба видалити, оскільки у цьому випадку L_{X_J} не містить конструктора сорту *v*. Прикладом такої операції є операція додавання *Add*. Якщо ж операція φ не визначена на u, її інтерпретатор треба визначити. Прикладом є операція ділення *Div*. Тому специфікація *Div* містить співвідношення $a/b = a!/b.$

Оптимізація інтерпретатора полягає у оптимізуючих перетвореннях умов та правих частин цих рівностей. Розглянемо правило виду $U_{X_J} \to L_{X_J} = R_{X_J}$, – частинний випадок специфікації (3.7). Права частина загального правила має вид $Form(t, R)$, де $t = f(a_1,...,a_m, B_1,..., B_m, P)$, $r_i = g_i(a_1,...,a_m, B_1,..., B_m, P), i = 1,..., k.$

При обчисленні частинного випадку, тобто при підстановці, визначеній рівностями (3.8), отримуємо співвідношення

$$t_J = f_J(a_1,...,a_m, C_1,..., C_m, P),\ r_{i,J} = g_{i,J}(a_1,...,a_m, C_1,..., C_m, P), i = 1,..., k.$$

Терми $t_J = f_J(a_1,...,a_m, C_1,..., C_m, P)$, $r_{i,J} = g_{i,J}(a_1,...,a_m, C_1,..., C_m, P)$, можна, взагалі кажучи, спростити перетвореннями в алгебрах аргументів. Ті ж самі перетворення відносяться до термів умови $U_{X_J}(t_J, R_J)$. У розглянутому прикладі права частина правила $a + c // d = (a * d + 1 * c)!/(1 * d)$ спрощується, тому правило має вид $a + c // d = (a * d + c)!/d.$ Заключне спрощення полягає у елімінації, якщо це можливо, функції *Form*: якщо терми $t_J = f_J(a_1,...,a_m, C_1,..., C_m, P)$, $r_{i,J} = g_{i,J}(a_1,...,a_m, C_1,..., C_m, P)$ задовольняють контекстній умові $\pi(t_J, R_J)$, функцію $Form(t_J, R_J)$ можна замінити на конструктор сорту $\tau(t_J, R_J)$. У нашому прикладі права частина правила $a + c // d = (a * d + c)!/d$ ще спрощується: $a + c // d = (a * d + c) // d$, оскільки терми $a*d + c, d$ задовольняють контекстній умові $GCD(a * d + c, d) = 1$ при $GCD(c, d) = 1.$

Метод виводу системи правил інтерпретатора алгебраїчної операції зі спрощенням умов та правих частин можна реалізувати у вигляді алгебраїчної програми, оскільки він спирається тільки на екваціональний вивід. Для автоматизації елімінації функції *Form* треба використовувати більш складні методи – системи доведення теорем над базовим сортом u.

3.3.3. Вивід алгебраїчних програм інтерпретаторів операцій у лінійних динамічних розширеннях

Приклад 3.10. Специфікації сорту *Polynom* та вивід обчислень з многочленами однієї змінної. Специфікації сорту *Polynom* визначають цей сорт як евклідову область та лінійне динамічне розширення *Monom*.

```
Sort Polynom::EuclideDomain;
Parameter Field Coef, Const Variable Argument;
Constructor{
Polynom P = Monom M ++ Polynom Q | Monom M // Конструктор сорту
0 ++ P = P,                  // Функція вкладення  PolynomToPolynom
M ++ 0 = M;                  // Функція вкладення  PolynomToMonom
LeadMon(P) = M,                    // Функції доступу
LeadCoef(P)= Cf(M),
Arg(P)= Arg(M),
Deg(P)= Deg(M);
Deg(M)> Deg(Q),              // Контекстна умова
Arg(M)= Arg(Q);
Form: M∈Monom, Q∈Polynom,
 0 ++ P = P, M ++ 0 = M,
 Arg(M) = Arg(Q), Deg(M)> Deg(Q), Cf(M)<>0
};
Operations
Add: Deg(a)==Deg(b)→ (a++A)+(b++B)=(a+b) !+ (A+B);
Sub: Deg(a)==Deg(b)→ (a++A)-(b++B)=(a-b) !+ (A-B);
Mult: // Polynom * Polynom → Polynom; Commutative
(a++A)*(b++B)=(a*b)++((a*B+A*b)+A*B);
Mult: // Coef * Polynom → Polynom; Commutative
c*(b++B)= c*b ++ c*B;
(b++B)*c = Form(c*b, c*B);
Div: (a++A)/b = a/b ++ A/b;
Pow: a^n = sqr(a^n div 2)*a^(n mod 2); // З сорту MiltSemiGroup
IntDiv:
Deg(P)==Deg(Q)→ P div Q = LeadCoef(P)/LeadCoef(Q),
Deg(P)< Deg(Q)→ P div Q = 0,
Deg(P)> Deg(Q)→ P div Q = LeadMon(P) div LeadMon(Q)++
                (P-(LeadMon(P)div LeadMon(Q))*Q div Q);
Mod: // З сорту EuclideDomain
P mod Q = P - (P div Q)*Q;
}
```

Далі буде показано, як виводяться специфікації операцій сорту *Polynom*. Ми також покажемо, що методи виводу специфікацій, розглянуті вище, приводять до обґрунтованих з математичного погляду систем інтерпретуючих правил.

Перш за все, звернемо увагу на два принципово різних методи визначення операцій. Операції *Add, Sub, Mult, Div* визначені у термінах конструкторів

операндів. Такі визначення операцій ми будемо називати *конструктивними*. Зауважимо, що у конструктивних визначеннях операцій істотну роль грають функції доступу. Особливо яскраво це демонструє визначення операції *IntDiv* (ділення с остачею). Операцію *Mod* визначено без використання конструктора – за допомогою тільки операцій сигнатури сорту *Polynom*. Такі визначення операцій ми будемо називати *абстрактними*. Оскільки цю сигнатуру успадковано від абстрактного сорту *EuclideDomain*, правильно було б і специфікацію операції *Mod* навести саме у визначенні цього сорту. До речі, оскільки алгоритм Евкліда формулюють у термінах цієї операції, у сорті *EuclideDomain* треба визначити і алгоритм Евкліда. А операцію *Pow* треба визначити ще раніше – у специфікаціях сорту *MiltSemiGroup* (див рис. 3.5, сорт *Мультиплікативна напівгрупа*).

Оскільки конструктор сорту визначений рекурсивно, алгебра *Polynom* є послідовністю (3.4) вкладених алгебр (башти алгебр), яка починається з алгебри *Monom* (мономи однієї змінної):

$$Mon = Pol_0 \subset Pol_1 \subset ... \subset Pol_k \subset ... \qquad (3.9)$$

Інтерпретація алгебри Pol_i: ця алгебра є множиною поліномів степеня i. Тоді Pol_i є векторними просторами розмірності $i+1$. При такій інтерпретації степінь поліному визначає його індекс у послідовності. Тому операцію додавання *Add* визначено трьома рівностями, перша з яких визначає правило додавання, якщо обидва операнди належать одній алгебрі, інші дві – різним:

$$a,b \in Pol_i; a \in Pol_i, b \in Pol_j, i < j; \; a \in Pol_i, b \in Pol_j, i > j$$

Таким чином, розширення (3.9) є розширенням векторних просторів. Оскільки у векторному просторі визначені адитивні операції та операції множення та ділення на скаляр, правила інтерпретації цих операцій виводяться з їх специфікацій цілком аналогічно вже розглянутому випадку статичного розширення алгебр:

$$\dim(Pol_i) = i+1, \; \dim(Pol_{i+1}) = i+2, \; Pol_i \subset Pol_{i+1},$$

$$V \in Pol_{i+1} \to V = a++A, A \in Pol_i; 0++A = A, \; a++0 = a.$$

Основне правило додавання є правилом покоординатного додавання векторів:

`Deg(a)==Deg(b)` $\rightarrow$ `(a++A)+(b++B)=(a+b)!+(A+B)`.

Похідні правила отримуються з представлень векторів меншої розмірності як векторів більшої розмірності з нульовою першою координатою. Спрощуючи праві частини правил, отримуємо:

`(0++A)+(b++B)=(0+b)!+(A+B)`, `(a++A)+(0++B)=(a+0)!+(A+B)`.

Враховуючи умови, отримуємо систему правил інтерпретації операції додавання, яка враховує вкладення $0++P=P$:

```
Deg(a)== Deg(b)→ (a++A) + (b++B)=(a+b) !+ (A+B),
Deg(A)< Deg(b) → A + (b++B) = b !+ (A+B),
Deg(a)> Deg(B) → (a++A) + B = a !+ (A+B),
```

Виклики інтерпретатора конструктора «++» потрібні тому, що виведені правила ще не враховують другу з умов вкладення `M++0 = M`. Тому кожне з цих правил треба ще перетворити: при `A = 0` з

```
Deg(a)==Deg(b)→(a++A)+(b++B)=(a+b)!+(A+B)
```

отримуємо:

```
Deg(a)==Deg(b)→ a+(b++B)=(a+b)!+B.
```

Аналогічно, при `B = 0` отримуємо:

```
Deg(a)==Deg(b)→(a++A)+b=(a+b)!+A.
```

Нарешті, при `A = 0`, `B = 0` отримуємо:

```
Deg(a)==Deg(b)→a+b=a!+b.
```

Це правило визначає частинну операцію додавання на сорті Monom. Тому його треба включити до специфікації цього сорту. Для другого правила при B = 0 отримуємо:

```
Deg(A)<Deg(b)→A+b = b !+ A.
```

Виклик інтерпретатора «++» є зайвим. Тому

```
Deg(A)<Deg(b)→A+b = b++A.
```

Аналогічно для третього правила:

```
Deg(a)>Deg(B)→ a+B=a ++ B.
```

Отже, для операції *Add* сорту *Polynom* отримуємо таку систему правил інтерпретації:

```
Deg(a)== Deg(b)→ (a++A) + (b++B) = (a+b) !+ (A+B),
Deg(a)== Deg(b)→ a + (b++B)=(a+b) !+ B,
Deg(a)== Deg(b)→ (a++A) + b = (a+b) !+ A,
Deg(A)< Deg(b) → A + (b++B) = b !+ (A+B),
Deg(A)< Deg(b) → A + b = b ++ A,
Deg(a)> Deg(B) → (a++A) + B = a !+ (A+B),
Deg(a)> Deg(B) → a + B = a ++ B;
```

Крім того, на сорті *Monom* визначаємо частинну операцію Add:

```
a$x + b$x = (a + b)$x.
```

Отже:

1. Специфікації сорту *Polynom* визначають динамічне розширення векторного простору.

2. Абстрактні операції мають були виключені зі специфікацій *Polynom* і віднесені до специфікацій відповідних абстрактних алгебр.

3. Конструктивні операції сигнатури векторного простору визначаються основним випадком. Рівність, що визначає операцію в основному випадку, є умовною. Умова визначає належність усіх операндів одному члену послідовності (3.9). У цьому випадку результат операції також належить цьому члену, але у навіть у загальному випадку він може бути спрощений співвідношеннями вкладення. Частинні випадки виводяться методами виводу статичних розширень.

4. Оскільки для сорту *Polynom* існує два співвідношення вкладення, вивід повної системи правил здійснюється послідовно: спочатку для першого, а потім – для другого.

5. Операції множення та неповного ділення специфікуються окремо як додаткові операції на векторному просторі *Polynom*.

```
Sort Polynom::EuclideDomain, VectorSpace;
Parameter Field Coef, Const Variable Argument;
Constructor{
Polynom P = Monom M ++ Polynom Q | Monom M // Конструктор сорту
0 ++ P = P,                 // Функція вкладення  PolynomToPolynom
M ++ 0 = M;                 // Функція вкладення  PolynomToMonom
LeadMon(P) = M,                   // Функції доступу
LeadCoef(P)= Cf(M),
Arg(P)= Arg(M),
Deg(P)= Deg(M);
Deg(M)> Deg(Q),             // Контекстна умова
Arg(M)= Arg(Q);
Form: M∈Monom, Q∈Polynom,
Arg(M) = Arg(Q), Deg(M)> Deg(Q)
};
Operations
Add: Deg(a)==Deg(b)  → (a++A)+(b++B)= Form(a+b, A+B),
Sub: Deg(a)==Deg(b)  → (a++A)-(b++B)= Form(a-b, A-B),
Mult:
S*(b++B)= Form(S*b, S*B);
(b++B)*S = Form(S*b, S*B);
Div: (a++A)/Scal = Form(a/S, A/S);
Mult: (a++A)*(b++B)=Form(a*b, (a*B+A*b)+ A*B);
IntDiv:
```

```
    Deg(P)==Deg(Q) → P div Q = LeadCoef(P)/LeadCoef(Q),
    Deg(P)< Deg(Q) → P div Q = 0,
    Deg(P)> Deg(Q) → P div Q =
    Form(LeadMon(P)div LeadMon(Q),(P-(LeadMon(P)div LeadMon(Q))*Q
div Q);
```

У подальшому, для прикладу 3.13, нам потрібні будуть усі правила інтерпретації операції множення. Тому виведемо правила – частинні випадки *Mult*:

```
A*(b++B)= A*b + A*B, (a++A)*B = a*B + A*B;
```

Однак, якщо для першого правила множник A, а для другого правила – множник B є поліномами, тобто `A` має вид `a1++A1`, `B` має вид `b1++B1`, ці частинні випадки нічим не відрізнятимуться від загального правила. Тому у частинних правилах A та B є мономами. Отже, хоча це і не обов'язково, краще змінити великі букви на маленькі

```
Mult:
    (a++A)*(b++B)= a*b ++ ((a*B+A*b)+ A*B),
    a*(b++B)= a*b + a*B,
    (a++A)*B = a*b + A*b;
```

3.3.4 Вивід алгебраїчних програм інтерпретаторів операцій в бінарних динамічних розширеннях

Розглянемо бінарне динамічне розширення алгебри *Bool* – алгебру логіки *BoolAlg.* Ця алгебра є також розширенням базового сорту *Variable,* оскільки елементи цього сорту – латинські букви – інтерпретуються як логічні формули. Ми розглянемо специфікації сорту *BoolAlg* і покажемо, що вивід інтерпретаторів логічних операції здійснюється тими ж методами.

Елементами сорту *BoolAlg* є формули логіки висловлювань багатьох змінних. Нехай $F(x_1, x_2, ..., x_n)$ – довільна формула формули логіки висловлювань від n змінних. Позначимо через O, I логічні значення відповідно *істина* та *хибність*. Тоді

$$F(x_1, x_2, ..., x_n) = x_n \,\&\, F(x_1 x_2, ..., x_{n-1}, I) \vee \neg x_n \,\&\, F(x_1, x_2, ..., x_{n-1}, O).$$

Якщо позначити

$$A(x_1, ..., x_{n-1}) = F(x_1, ..., x_{n-1}, I), B(x_1, ..., x_{n-1}) = F(x_1, ..., x_{n-1}, O),$$

отримаємо представлення

$$F(x_1, x_2, ..., x_n) = x_n \,\&\, A(x_1, ..., x_{n-1}) \vee \neg x_n \,\&\, B(x_1, ..., x_{n-1}) \qquad (3.10)$$

Здійснивши послідовно такі ж перетворення формул *A, B* відносно змінних $x_{n-1},...,x_1$, отримаємо рекурсивне представлення формули логіки висловлювань. Насправді, через $BoolAlg_m$ позначимо множину формул логіки висловлювань від змінних $x_1,...,x_m$. Тоді

$$BoolAlg_n = \{F : F = x_n \,\&\, A \vee \neg x_n \,\&\, B, A, B \in BoolAlg_{n-1}\}$$

Отже, сорт *BoolAlg* є об'єднанням зростаючої послідовності алгебр $BoolAlg_m$:

$$BoolAlg_0 = Bool,\ BoolAlg_0 \subset BoolAlg_1 \subset ... \subset BoolAlg_m \subset \quad (3.11)$$
$$BoolAlg = \cup BoolAlg_m$$

Розширення такого типу є бінарними (означення 3.15). Зауважимо, що формула (3.10) задає канонічну форму формули алгебри висловлювань. Незважаючи на те, що ця канонічна форма менш відома, ніж ДКНФ та ДДНФ, вона має властивості, що дозволяють достатньо просто визначити обчислення. Дійсно, позначимо

$$BF(A,B,x) \overset{df}{=} x \,\&\, A \vee \neg x \,\&\, B.$$

Тоді

$$BF(A_1,B_1,x) \,\&\, BF(A_2,B_2,x) = BF(A_1 \,\&\, A_2, B_1 \,\&\, B_2, x),$$
$$BF(A_1,B_1,x) \vee BF(A_2,B_2,x) = BF(A_1 \vee A_2, B_1 \vee B_2, x),$$
$$\neg BF(A,B,x) = BF(\neg A, \neg B, x).$$

Таким чином, основні логічні операції виконуються поаргументно! Нарешті, легко перевірити, що функція вкладення визначається рівністю

$$BF(A,A,x) = A.$$

Цю канонічну форму формули алгебри висловлювань називають *рекурсивною нормальною формою* (РНФ). Наведемо далі специфікації сортів *Bool*, *BoolAlg*.

Приклад 3.11. Специфікації сорту *BoolAlg* та вивід обчислень логічними формулами багатьох змінних. Специфікації сорту *BoolAlg* визначають цей сорт як булеву алгебру – нащадок абстрактної алгебри *Bool* та бінарне динамічне розширення *Bool*.

```
Sort Bool;
 // Абстрактна алгебра висловлювань: сигнатура, аксіоми, операції з константами.
Axioms
 ~O = I,              ~I = O,
 A&O = O,             A&I = A,
 A|O = A,             A|I = I,
```

```
 A&~A  = O,           A|~A = I,
 (A&B)&C = A&(B&C),   (A|B)|C = A|(B|C),
 A&B = B&A,           A|B = B|A,
 A&A = A,             A|A= A,
 A&(B|C) = A&B|A&C,   A|(B&C)= (A|B)&(A|C),
 ~(A&B) = ~A|~B,      ~(A|B) = ~A&~B, // правила де Моргана
           ~~A = A;
Operations
 Con: A&O = O, A&I = A;
 Dis: A|O = A, A|I = I;
 Neg: ~O = I, ~I = O;
}

Sort BoolAlg::Bool;
Constructor{
 BoolAlg A = BF(BoolAlg A, BoolAlg B, Variable x)|Bool A;
 BF(A, A, x) = A;          // Функція вкладення  BoolAlgToBoolAlg
 Arg(BF(A, B, x)) = x,
 Left(BF(A, B, x))= A,
 Right(BF(A, B, x))= B;
 x > Arg(A), x > Arg(B);    // Контекстна умова
 Form:  A, B ∈ BoolAlg, x > Arg(A), x > Arg(B)
};
Operations
Con: BF(A1,B1,x)& BF(A2,B2,x)= BF(A1&A2,B1&B2,x);
Dis: BF(A1,B1,x)|BF(A2,B2,x)= BF(A1|A2,B1|B2,x);
Neg: ~BF(A,B,x)= BF(~A,~B,x);
```

Для виводу системи правил інтерпретації, як і у попередньому прикладі, використовуємо умову вкладення, покладаючи `A1=B1` і позначивши `A1` через `A`. Отримаємо :

```
x > Arg(A)→ A&BF(A2,B2,x)= BF(A&A2,A&B2,x).
```

Аналогічно при `A2=B2`, позначивши `A2` через `A`, отримаємо:

```
x > Arg(A) → BF(A1,B1,x)& A = BF(A1&A,B1&A,x).
```

Отже, система правил інтерпретації операції диз'юнкції має вид:

```
BF(A1,B1,x)& BF(A2,B2,x)= BF(A1&A2,B1&B2,x),
x > Arg(A)→ A&BF(A2,B2,x)= BF(A&A2,A&B2,x),
x > Arg(A)→ BF(A1,B1,x)&A = BF(A1&A,B1&A,x).
```

Зауважимо, що у специфікаціях операцій не використовується функція *Form*. Справа у тому, що специфікації цієї функції уже виконуються для правих частин наданих визначень логічних операцій. Наприклад, у визначенні операції *Dis* права частина `BF(A1|A2,B1|B2,x)` відповідає умовам `x>Arg(A1|A2)`, `x>Arg(B1|B2)`, оскільки для лівої частини визначення виконуються нерівності

```
x>Arg(A1), x>Arg(B1), x>Arg(A2), x>Arg(B2).
```

3.4. Верифікація інтерпретаторів алгебраїчних операцій

У попередньому пункті ми показали, що поняття конструктивного розширення алгебр дозволяє здійснювати логічно обґрунтований вивід (синтез) інтерпретуючих правил операцій та їх оптимізацію, спираючись на *базові правила, або* правила «загального випадку», які мають бути визначеними у специфікаціях відповідного сорту. Таким чином, *проблема верифікації алгебраїчних обчислень зводиться до проблеми обґрунтування правильності правил «загального випадку».* Які властивості притаманні «правильним» специфікаціям операцій? Як відрізняти «правильні» означення від «неправильних»? Для відповіді на ці питання маємо перевірити виконання наступних умов:

1. Правила специфікації алгебраїчних операцій, визначених конструктивно в специфікаціях алгебри, мають задовольняти системі аксіом цієї алгебри.

2. Результат виконання операції має відповідати синтаксичним означенням та контекстним умовам конструктора сорту.

Далі покажемо, що ключову роль в цих задачах грають алгоритми індуктивних доведень [53-56].

3.4.1. Приклади верифікації інтерпретаторів операцій

Приклад 3.12. Верифікація сорту BoolAlg.
Розглянемо аксіому сорту *Bool*, наприклад, правило де Моргана

$$\sim(A\&B) = \sim A|\sim B.$$

Підставимо замість A, B їх конструктивні означення:

```
A = BF(A1,B1,x), B = BF(A2,B2,x).
```

Отримаємо:

```
~(BF(A1,B1,x)& BF(A2,B2,x)) = ~BF(A1,B1,x)|~BF(A2,B2,x)
```

Виконаємо обчислення у лівій та правій частинах рівності та випишемо відповідні контекстні умови:

```
BF(~(A1&A2),~(B1&B2),x) = BF(~A1|~A2,~B1|~B2,x),        (3.12)
Arg(x)>Arg(~(A1&A2)), Arg(x)>Arg(~(B1&B2)),             (3.13)
Arg(x)>Arg(~A1|~A2)), Arg(x)>Arg(~B1|~B2)).             (3.14)
```

Відповідно до означення конструктора *BF(A,B,x)* , рівність (3.12) означає, що її аргументи або синтаксично рівні, або зводяться до синтаксично рівних

функцією вкладення $BF(A,A,x)=A$. Оскільки інших контекстних умов немає, рівності аргументів (3.12) мають бути тотожностями алгебри висловлювань

```
~(A1&A2)≅~A1|~A2,~(B1&B2)≅~B1|~B2.
```

Висновок: перевірка виконання аксіоми де Моргана на елементах сорту зводиться до перевірки її виконання на аргументах конструктора сорту. Тому доведення має здійснюватися індукцією за індексами башти алгебр (3.11). Крок індукції полягає у доведенні (умовної тотожності):

```
~(A1&A2)≅~A1|~A2,~(B1&B2)≅~B1|~B2→
~(BF(A1,B1,x)&BF(A2,B2,x))= ~BF(A1,B1,x)|~BF(A2,B2,x),
```

та доведенні нерівностей (3.12)-(3.14).

Приклад 3.13. Верифікація сорту *Polynom*.

Розглянемо одну з аксіом сорту *Polynom*, наприклад, аксіому дистрибутивності $x(y+z)=xy+xz$. Як і у попередньому прикладі, підставимо замість x,y,z їх конструктивні означення :

```
x = a++A, y =b++B, z = c++C.
```

Отримаємо:

```
(a++A)*((b++B)+(c++C)) = (a++A)*(b++B)+(a++A)*(c++C).
```

Виконаємо обчислення за основними правилами зі специфікації операцій сорту *Polynom* у лівій та правій частинах рівності та випишемо відповідні контекстні умови :

```
a(b+c)++((a(B+C)+A(b+c))+A(B+C))=
(ab+ac)++((aB+Ab)+AB)+((aC+Ac)+AC)),
Deg(b)=Deg(c), Deg(a)>Deg(A), Deg(b)>Deg(B), Deg(c)>Deg(C).
```

Звідки отримаємо дві рівності та нерівність, які треба довести:

```
a(b+c)=ab+ac,                                                    (3.15)
(a(B+C)+A(b+c))+A(B+C)=((aB+Ab)+AB)+((aC+Ac)+AC));               (3.16)
Deg(a(b+c))>Deg(((a(B+C)+A(b+c))+A(B+C))).                       (3.17)
```

Розглянемо постановку задачі доведення нерівності (3.17). Ця нерівність є наслідком правил виконання операцій додавання та множення. Якщо ці правила правильно оперують з степенями, вона виконується автоматично. Тому задача доведення нерівностей, які формулюються як контекстні умови правильності аксіом, зводиться до задач доведення правильності умов у правих частинах правил інтерпретації операцій. У нашому випадку треба доводити такі умовні нерівності:

Для основного правила операції `Add`:

```
Deg(a)=Deg(b),Deg(a)>Deg(A),Deg(b)>Deg(B)→Deg(a+b)> Deg(A+B).
```

Для основного правила операції `Mult`:

```
Deg(a)>Deg(A), Deg(b)>Deg(B)→Deg(a*b)>Deg((a*B+A*b)+ A*B).
```

Перша з них є наслідком загального правила:

```
Deg(P+Q) = Max(Deg(P), Deg(Q)), якщо LeadMon(P+Q)≠0.
```

Отже, якщо позначити

```
Deg(a)=k, Deg(b)=l, Deg(A)=k1, Deg(B)=l1, k=l,
        k>k1, l>l1 → Max(k,l) > Max(k1,l1).
```

Аналогічно, для другої загальним правилом є

```
Deg(P*Q)=Deg(P)+ Deg(Q).
```

При тих же позначеннях

```
k=l, k>k1, l>l1 → k+l > Max(k1+l,l1+k,k+l).
```

Висновок: доведення правильності контекстних умов в окремій операції зводиться до доведення умовних нерівностей, які формулюються для цієї аксіоми на основі правил обчислення степенів поліномів – результатів виконання операцій сорту. При цьому треба перевіряти не тільки основні правила інтерпретації, а і похідні.

Розглянемо тепер постановку задачі доведення рівностей. Рівності (3.15), (3.16), як і у прикладі 3.12, мають місце с точністю до співвідношень вкладення сорту: $M{+}{+}0 = M$, $0{+}{+}P = P$. Отже, вони є тотожностями алгебри многочленів. На відмінність від попереднього прикладу, рівність (3.16) не є аксіомою . Перша з них – це дистрибутивний закон для базового сорту, друга – тотожність у алгебрі поліномів, яка доводиться з аксіом сорту – дистрибутивності, комутативності та асоціативності. Таким чином, доведення може здійснювати методом індукції у такій формулировці:

1. *База індукції*: Для елементів a, b, c, A, B, C сорту *Monom* виконуються усі аксіоми сорту *Polynom*.

2. *Припущення індукції*: Для елементів A, B, C сорту *Polynom* та елементів a, b, c сорту *Monom* виконуються усі аксіоми сорту *Polynom*.

2. *Крок індукції*: Для елементів $X = a{+}{+}A$, $Y = b{+}{+}B$, $Z = c{+}{+}C$ сорту *Polynom* також виконуються усі аксіоми сорту *Polynom*.

Висновок індукції: На сорті *Polynom* виконуються усі аксіоми сорту *Polynom*.

Зауважимо, що перевірка бази індукції потребує визначення операції додавання на сорті *Monom*. Для нашого прикладу можна вважати, що Сорт *Monom* визначається як статичне розширення сорту *Degree* конструктором

$$Monom\ M\ =\ (Field\ a)\$(Degree\ D),\ 1\$D\ =\ D,$$

На цьому сорті операція Add є частковою: $a\&A\ +\ b\$A\ =\ (a+b)\$A$.

Висновок: перевірка виконання окремої аксіоми (наприклад, дистрибутивності) на елементах сорту *Polynom* зводиться до перевірки виконання усієї системи аксіом на аргументах конструктора сорту. Тому доведення має здійснюватися у формі доведення усієї системи аксіом сорту індукцією наведеної форми для тільки для основних правил інтерпретації операцій.

Приклад 3.14. Верифікація сорту *Rat*.

Розглянемо, як приклад, задачу перевірки виконання аксіоми дистрибутивності для сорту *Rat*. Як ми показали вище, на множині $Int \times Nat$ має бути визначена конгруенція, яка виділяє пари аргументів (чисельник, знаменник) конструктора сорту *Rat*. Ця конгруенція має вид

$$(a = p_1 // q_1,\ b = p_2 // q_2,\ a \sim b \leftrightarrow p_1 * q_2 = p_2 * q_1)\ .$$

Отже, для елементів сорту Rat має місце співідношення

$$(p!/q = r//s)\&(p*s = r*q)\&(GCD(r,s) = 1) \qquad (3.18)$$

Розглянемо доведення аксіоми дистрибутивності. Нехай

$$a(b+c)\ =\ ab+ac,\ a = p_1//q_1,\ b = p_2//q_2,\ c = p_3//q_3.$$

Підставивши представлення a, b, c в аксіому та здійснивши обчислення за правилами з специфікацій *Rat*, отримаємо:

$$p_1(p_2q_3 + q_2p_3)!/(q_1q_2q_3) = (q_1q_3p_1p_2 + q_1q_2p_1p_3)!/(q_1q_2q_1q_3)$$

Для спрощення виразів введемо позначення:

$$A\ =\ p_1(p_2q_3 + q_2p_3),B\ =\ q_1q_2q_3,\ C\ =\ q_1q_3p_1p_2\ +\ q_1q_2p_1p_3,\ D\ =\ q_1q_2q_1q_3.$$

Тоді верифікація аксіоми дистрибутивності зводиться до перевірки тотожності $AC = BD$ над сортом *Int* та доведенні (3.18) на сорті *Rat*. Доведення тотожності рівності верифікує дану аксіому, доведення співвідношення (3.18) верифікує інтерпретатор конструктора сорту.

3.4.2. Метод верифікації інтерпретаторів алгебраїчних операцій

Розглянемо алгебри $A =< S_A, U_A, T_A, \Sigma_A, Ax_A, I_A >$, $B =< S_B, U_B, T_B, \Sigma_B, Ax_B, I_B >$.

Узагальнимо метод верифікації системи алгебраїчних програм – інтерпретаторів операцій сорту v з його специфікацій за умови, що алгебру A_v визначено як *статичне розширення* алгебри A_u сорту $u: A_u \subset A_v$. Нехай $A \overset{df}{=} A_u$, $B \overset{df}{=} A_v, A \subset B.$ Далі ми будемо використовувати позначення п. 3.2.2. Специфікації сорту v визначають:

Конструктор сорту: $v = \tau(u, W)$.

Обов'язковим аргументом τ є сорт u.

Контекстна умова: $\pi(u, W)$.

Умова вкладення: $\rho(u, W)$.

Функція вкладення: $\rho(u, W) \to \tau(u, W) = u$.

Предикат $\rho(u, W)$ є функціональним: для кожного значення u існує єдине значення P таке, що $\rho(u, W)$. Цю функцію ми будемо позначати через $F\rho$: $\rho(u, W) = (F_p(u) = W)$.

Інтерпретатор конструктора сорту: $!\tau(x, P) = y$.

Конгруенція конструктора сорту: Конгруенція Θ на множині $S_u \times S_{w_1} \times ... \times S_{w_k} : (x, p_1, ..., p_k) \overset{\Theta}{\approx} (x', p_1', ..., p_k')$ визначає умову на аргументах конструктора сорту: якщо $\tau(x, P) = \tau(x', P')$, то $(x, P) \overset{\Theta}{\approx} (x', P')$. Неважко побачити, що конструкція елементу сорту є канонічною формою. Тому ця конгруенція визначає класи еквівалентності елементів $S_u \times S_{w_1} \times ... \times S_{w_k}$, які мають єдину канонічну форму.

Сигнатури операцій сортів u, v: $\Sigma_u = < \varphi_1, ... \varphi_m >$; $\Sigma_v = < \varphi_1, ... \varphi_m, \psi_1, ..., \psi_l >$. Ми вважаємо, що сигнатура операцій розширення B є розширенням сигнатури операцій базової алгебри A.

Аксіоми сортів u, v: $AX_u = < A_1, ... A_s >$; $AX_v = < B_1, ..., B_{s'} >$. Ми вважаємо, що система аксіом розширення B є розширенням системи аксіом базової алгебри A.

Специфікації операції $\varphi_1, ... \varphi_m$ задані системами рівностей, аналогічними (3.7).

Верифікація інтерпретаторів операцій алгебри B полягає у:

1. Доведенні тотальної коректності інтерпретатора функції $!\tau(x, P)$.
2. Доведенні тотожностей над алгеброю A, які є наслідками аксіом AX_B та конгруенції Θ конструктора алгебри B.

Доведення часткової коректності інтерпретатора $!\tau(x, P)$ спирається на його представлення у вигляді суперпозиції функцій побудування канонічної форми аргументу (x, P) як елементу носія алгебри B та застосуванні вкладення до цієї форми $\rho(x, P) \to (!\tau(x, P) = x)$.

Позначимо через τ_{can} функцію, яка будує аргумент (x,P) як елемент носія алгебри B, а через τ_{red} – функцію, яка здійснює вкладення $\rho(x,P)\to(!\tau(x,P)=x)$. Тоді $!\tau(x,P)=\tau_{red}(\tau_{can}(x,P))$. Частинну коректність $!\tau(x,P)$ можна довести, довівши імплікації

$$(\tau_{can}(x,P)=(x',P'))\to((x,P)\overset{\Theta}{\sim}(x',P'))\,\&\,\pi(x',P')) \tag{3.19}$$

$$\rho(x,P)\,\&\,(\tau_{red}(x,P)=x)\to((x,P)\overset{\Theta}{\sim}(x,F_\rho(x))\,\&\,\pi(x,F_\rho(x))) \tag{3.20}$$

Означення 3.16 . Розширення алгебри $A\subset B$ назвемо прямим, якщо функція τ_{can} є тотожною.

Елемент носія прямого розширення B є довільним елементом $S_u\times S_{w_1}\times\ldots\times S_{w_k}$. Конгруенція Θ є одиничною. Таким чином, при прямих розширеннях доводити треба лише коректність вкладення $A\to B$.

Розширення $A_i\subset A_{i+1}$ прикладів 3.12, 3.13 є прямими. Більшість розширень, які зустрічаються у цій роботі, є прямими. Розширення $Int\to Rat$ (приклад 3.14)) є прикладом розширення, яке не є прямим.

Розглянемо тепер *доведення тотожностей*. Тотожності над A, які є наслідками аксіом AX_B та конгруенції Θ отримуються таким чином: Якщо рівність $B_i\overset{df}{=}(L(x_1,\ldots,x_n,P)=R(x_1,\ldots,x_n,P))\in AX_B$, причому змінні $x_1,\ldots,x_x$ визначені над B, підстановки $\tau(u_i,P_i)$ замість x_i визначають рівності над $S_u\times S_{w_1}\times\ldots\times S_{w_k}$. Формальне виконання інтерпретаторів операцій алгебри B у лівій та правій частинах рівності $L(\tau(u_1,P_1),\ldots,\tau(u_n,P_n),P)=R(\tau(u_1,P_1),\ldots,\tau(u_n,P_n),P)$ приводить до рівності $\tau(f,G_1,,,G_k)=\tau(f',G_1',\ldots,G_k')$, де $f=f(u_1,..,u_n,P_1,..,P_n,P)$, $f'=f'(u_1,..,u_n,P_1,..,P_n,P)$, $G_i=G_i(u_1,..,u_n,P_1,..,P_n,P)$, $G_i'=G_i'(u_1,..,u_n,P_1,..,P_n,P)$, $i=1,..,n$. Отже, має місце еквівалентність

$$(f,G_1,,,G_k)\overset{\Theta}{=}(f',G_1',\ldots,G_k'). \tag{3.21}$$

Це співвідношення залежить від змінних $u_1,..,u_n,P_1,..,P_n,P$. Будемо вважати, що конгруентність Θ задано системою тотожних рівностей $V_j=W_j$, $j=1,..,e.$ Тоді верифікація інтерпретаторів операцій алгебри B полягає у доведенні тотожностей

$$V_j(u_1,..,u_n,P_1,..,P_n,P)=W_j(u_1,..,u_n,P_1,..,P_n,P),\ j=1,..,e. \tag{3.22}$$

над базовими сортами $u,w_1,\ldots,w_k$. Алгоритм доведення (3.22) має спиратися на аксіоми базових сортів. Отже, цей алгоритм існує, якщо проблема

тотожностей над базовими алгебрами є алгоритмічно розв'язуваною.

Якщо розширення $A \subset B$ є прямим, сукупність тотожностей (3.22) є сукупністю "по координатних" тотожностей. Тому конгруентність (3.21) визначає тотожності $f = f', G_i = G_i', i = 1,..,k$.

Узагальнимо тепер метод верифікації системи інтерпретаторів операцій сорту v з його специфікацій, якщо алгебру A_v є *динамічним розширенням* алгебри A.

Нехай $A_0 = A,\ A_{i+1} = \tau(A_0, A_i),\ B = \bigcup_{i=0}^{\infty} A_i$ – специфікація конструкції лінійного динамічного розширення $A \subset B$, $x \in A_0,\ X \in A_i, \rho(x, X) \to \tau_{red}(x, X) = X$ –вкладення, визначені для елементів A_i цього розширення. Як і для статичних розширень, будемо вважати, що система аксіом B є розширенням системи аксіом базової алгебри A. Це означає, що цій системі мають задовольняти усі алгебри A_i. Тоді:

- верифікація функції $!\tau$ здійснюється індукцією по індексу i та полягає у доведенні імплікацій (3.19), (3.20);

- верифікація інтерпретаторів операцій на B за допомогою аксіом здійснюється індукцією по індексу i та полягає у доведенні тотожностей (3.22) над алгебрами A_{i+1} за індуктивним припущенням, що усі аксіоми доведені як тотожності алгебри A_i. Якщо алгоритм доведення тотожності на $A_0 = A$ спирається тільки на аксіоми A, індукція дозволяє розширити цей алгоритм на B.

Насправді особливістю використання розширень як методів побудування алгебр є те, що на вже побудованій алгебрі B визначаються нові операції. Наприклад, сорт *Int* розширюється до *Rat* з метою визначення операції ділення (приклад 3.12).

Сорт *Polynom* розширюється як векторний простір. Таким чином, ми розглядаємо сорт *Monom* як одновимірний векторний простір, а крок розширення полягає у додаванні нової координати. У цьому прикладі A_i є i-вимірним векторним простором. Операції множення на скаляр, додавання та віднімання та усі аксіоми векторного простору підпадають під розглянутий нами метод. Але операція множення многочленів та усі аксіоми, які містять символ операції множення, мають бути розглянуті окремо.

Нехай $P, Q, R = P*Q \in Polynom,\ \deg P = i, \deg Q = j,\ A_k = \{p : \deg p \le k\}$. Тоді $\deg R = i + j$. Отже, якщо $P \in A_i, Q \in A_j, P*Q \in A_{i+j}$. Верифікація операції множення методом математичної індукції, розглянута вище (3.15), (3.16) зводить доведення аксіом на A_{i+j} до доведення тотожностей на A_{i+j-1}. Таким чином, базис індукції має містити доведення аксіом на базових алгебра, тобто на мономах. Наприклад, у (3.16) треба вважати, що $a, A, b, B, c, C \in Monom$.

Узагальнимо ці міркування для довільних алгебр. Метод індукції ми змогли застосувати, оскільки степінь добутку є монотонно зростаючою функцією степенів множників. Очевидно, це можна робити і у загальному випадку. Ми сформулюємо це для бінарних операцій.

Нехай $A_0 = A,\ A_{i+1} = \tau(A_0, A_i),\ B = \bigcup_{i=0}^{\infty} A_i$ та $\varphi(a,b)$ – бінарна операція, визначена на B. Тоді, якщо існує така монотонно зростаюча по кожному з аргументів функція $s(i, j)$, що для довільних $a \in A_i, b \in A_j\ \varphi(a,b) \in A_{s(i,j)}$, метод індукції зводить доведення аксіом B, які містять φ, до доведення цих аксіом для змінних, визначених на A.

3.5. Метод наслідування специфікацій алгебраїчних обчислень

У цьому параграфі ми розглянемо більш детально застосування методу наслідування у визначеннях багатосортних алгебр. Відзначимо, що наслідування, як і інші принципи об'єктно-оріентованого програмування, активно застосовуються в інших парадигмах програмування, зокрема, в логічному та алгебраїчному програмуванні [39, 42]. Однак наслідування, перш за все, є фундаментальним принципом побудови ієрархій математичних абстракцій, який активно використовувався задовго до виникнення програмування взагалі.

3.5.1. Ієрархії абстрактних алгебр

Метод наслідування, перш за все, застосовується на "абстрактному рівні" для визначення:

- сигнатур операцій,
- аксіом,
- інтерпретаторів операцій на константах,
- алгоритмів розв'язання прикладних задач, *незалежних від конструкторів відповідних сортів.*

На нашу думку, це – дуже корисний спосіб повторного використання коду. Сучасна абстрактна алгебра надає можливості зробити це у загальновідомих математичних поняттях та термінах (див. приклад 3.6, рис 1.2).

Отже, загальна ієрархія наслідування на "верхніх поверхах" є відомою структурою дефініцій абстрактних алгебр. Оскільки при цьому використовують різні позначення операцій та констант, деякі абстрактні алгебри присутні у ієрархії у різних формах позначень.

Найбільш очевидним є використання ієрархії абстрактних алгебр з метою визначення сигнатур операцій та систем аксіом. Тому слід розуміти, що метод наслідування навіть на абстрактному рівні виконує конструктивну роль.

Приклад 3.15. *Визначення абстрактної мультиплікативної підгрупи.*

На рис 1.2 алгебри *Напівгрупа*, *Комутативна Підгрупа* та *Група* позначені прямокутниками з пунктирними границями. Вони не несуть жодних конструктивних відомостей тільки тому, що знак напівгрупової операції не визначений. Кожна з інших алгебр цієї ієрархії визначає або принаймні одну нову аксіому, або нову операцію, або за допомогою множинного наслідування об'єднує ці визначення у єдине ціле. На абстрактному рівні визначаються також інтерпретації операцій – правила їх виконання у випадках, коли деякі з аргументів є виділеними константами.

Наведемо повну специфікацію *MultSemiGroup:*

```
Sort MultSemiGroup // мультиплікативна напівгрупа
Signature
 Mult(2): Internal, ()*();// внутрішня операція
 Pow(2): MultSemiGroup × Nat → MultSemiGroup, ()^();
 Sqr(1): Internal;
 1: Const;
Axioms
a*(b*c) = (a*b)*c,  1*a = a, a*1 = a,  a*b  =  b*a;
Operations
Mult: a*1 = a, 1*a = a; // множення на нейтральний елемент 1
Sqr: Sqr(a) = a*a; // операція піднесення до квадрату
Pow: // повне визначення операції підведення до натурального степеня
a^0=1, a^1=a, a^n = Sqr(a^(n div 2))*a^(n mod 2);
```

3.6. Метод ефективної реалізації алгебраїчних обчислень

Основним недоліком технологій символьних обчислень, реалізованих на основі систем переписуючи правил, є недостатня ефективність за часом. Тому такі системи програмування, як АПС, використовують для реалізації прототипів. Однак методологія та техніка алгебраїчного програмування є настільки привабливими, що актуальною становиться задача використання цієї технології, зокрема, системи АПС, як системи реалізації алгебраїчних задач у комерційних системах. Розглянемо один з підходів до цієї проблеми [51]. Цей підхід полягає у реалізації ключових алгоритмів алгебраїчних обчислень на нижньому рівні системи АПС (тобто, у вигляді внутрішніх функцій АПС, реалізованих мовою С). Ключові алгоритми алгебраїчних обчислень пропонується виявляти при аналізі специфікації відповідної багатосортної алгебри. Цей аналіз для математичних систем навчального призначення виявив такі алгоритми:

1. Обчислення у векторному просторі лінійних комбінацій.
2. Обчислення у групі мономів багатьох змінних.
3. Обчислення з дробами.

Векторний простір лінійних комбінацій. Векторний простір *LinSpace* формальних лінійних комбінацій над полем Coef визначений як множина лінійних комбінацій – лінійне динамічне розширення одновимірного

лінійного простору *LinMon* (див. приклад 3 до означення 3.14). Аналогічно, кільце многочленів однієї змінної, якщо його розглядати як векторний простір, є лінійним динамічним розширенням *Mon1*. (див. приклад 3.8). Ці приклади можна продовжувати.

Отже, можна визначити алгебру $W(X)$ (з ім.'ям *ALinComb*) абстрактних векторних просторів, де X – абстрактний лінійний порядок. Для будь-якого $x \in X$ $W(x)$ – лінійний простір, причому для будь-яких $x, y \in X$ лінійні простори $W(x)$, $W(y)$ є ізоморфними. Ізоморфізм визначається відображенням $x \to y$: $\varphi(w(x)) = w(y)$. Доступ до елемента $x \in X$ задається функцією $Ind : W(X) \to X$.

Далі треба побудувати лінійне динамічне розширення $L(W(X))$, використовуючи лінійний порядок X. Віртуальними методами (у термінології об'єктно-орієнтованого проектування) $L(W(X))$ є операції додавання і множення на скаляр, які визначають лінійний простір $W(x)$, відношення належності елементів одному і тому ж векторному просторі ($x = y$) та відношення лінійного порядку на X рис.3.8.

Алгебри *Polynom* та *LinSpace* визначаються як нащадки *ALinComb* з реалізацією віртуальних методів *AVectorSpace*. Операцію множення в алгебрі *Polynom* треба визначити або як власну операцію, або за допомогою наслідування *EuclidDomain*.

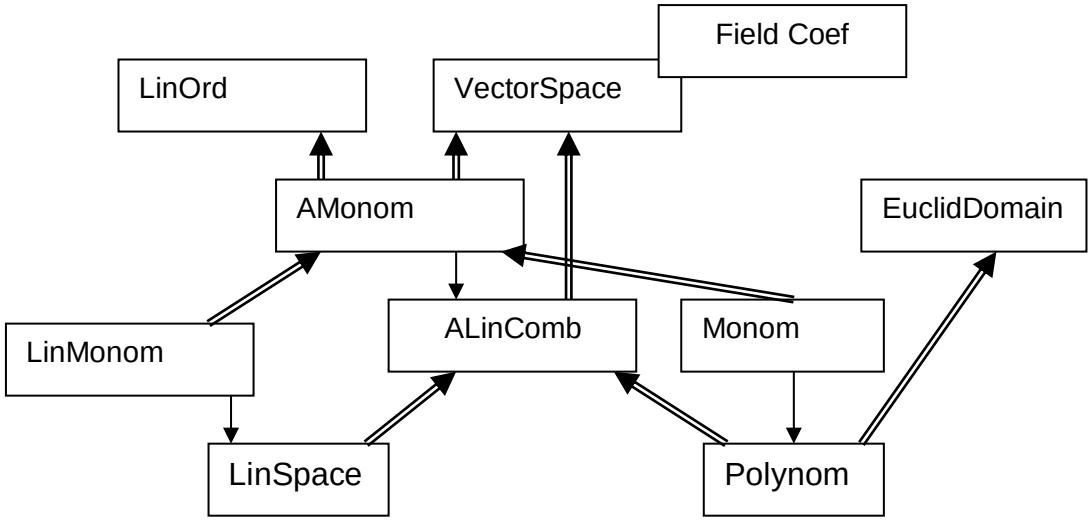


Рис. 3.8. Діаграма специфікацій векторних просторів лінійних комбінацій

```
Sort AMonom::VectorSpace, LinOrd;
     Parameter
     Field Coef;
     Signature
     Cf(1): AMonom -> Coef;
     Ind(1): AMonom -> LinOrd;
     Predicates
     Les: M < N = Ind(M) < Ind(N);
```

```
      Gre: M > N = Ind(M) > Ind(N);
      UnLes: M >= N = Ind(M) >= Ind(N);
      UnGre: M <= N = Ind(M) <= Ind(N);

Sort ALinComb::VectorSpace;
      Parameter AMonom;
      Constructor
      ALinComb P = {
      ALinComb P = AMonom M ++ ALinComb Q | AMonom M
      0 ++ P = P,
      M ++ 0 = M;
      LeadMon(P) = M,      LeadCoef(P)= Cf(M),  // Функції доступу
      Ind(M ++ Q) = Ind(Q); Ind(M)> Ind(Q);    // Контекстна умова
      Form: M ∈ AMonom, Q ∈ ALinComb,
      0 ++ P = P, M ++ 0 = M,
      Cf(M)<>0
      };
      Operations
      Add:
      Ind(a)== Ind(b) → (a++A)+(b++B)=(a+b) !+ (A+B),
      Ind(a)< Ind(b) → (a++A)+(b++B)= a++(A !+ (b++B)),
      Ind(a)> Ind(b) → (a++A)+(b++B)= b++((a++A) !+ B);
     // Інші правила виводяться методами лінійного динамічного розширення
      ScalMult: c$(a ++ A)= c$a ++ c$A;
     // Інші правила виводяться методами лінійного динамічного розширення

Sort LinMonom::AMonom;
      Parameter
      Atom Variable;
      Constructor
      LinMonom M = {
      (Coef c)$(Variable x)| Variable x;
      O$x = 0,
      1$x = x;
      Cf(a$x) = c;
      Ind(a$x) = x;
      };
      Operations
      Add: a&x + b$xA = (a+b)$x;
      CMult: c$(a$x) = (a*b)$x;
      Sort LinSpace::ALinComb;
      Parameter LinMonom;
```

Як бачимо, специфікація LinSpace полягає просто у визначенні відношень наслідування та розширення. *Усі алгоритми обчислень специфіковані* у LinSpace та LinMonom**.** Практично така ж схема використовується при специфікації кільця многочленів однієї змінної *Polynom*. Реалізація операції множення визначається у *Polynom* (див. приклад 3.10), а алгоритм Евкліда наслідується з *EuclidDomain*. Таким чином, на нижньому рівні реалізується *ALinComb*. Як показала практика, це прискорює обчислення у 3-5 разів.

Запропонована схема дозволяє також реалізувати як обчислення в алгебрі многочленів багатьох змінних, а також (як показано це нижче) обчислення в алгебрі тригонометричних многочленів та в інших алгебрах.

Звернемо тепер увагу на те, що обчислення в алгебрі мономів багатьох змінних по суті повторюють обчислення у векторному просторі лінійних комбінацій. Дійсно, лінійна комбінація має вигляд $L = a_1x_1 + a_2x_2 + ... + a_nx_n$, а степінь багатьох змінних – $M = x_1^{k_1}x_2^{k_2}...x_n^{k_k}$. Ізоморфізм цих алгебр очевидний: операція множення змінної на скаляр у лінійній комбінації відповідає піднесенню змінної до степеня у степені, а операція додавання – операції множення. Зауважимо, що алгебру мономів можна розглядати як групу. Отже, моному $M = b \cdot x_1^{k_1}x_2^{k_2}...x_n^{k_k}$ можна поставити у відповідність лінійну комбінацію $L = b + a_1x_1 + a_2x_2 + ..._a_nx_n$ з вільним членом. Тому на рівні C++ ці обчислення реалізуються в одному класі. Таким чином, отримуємо ефективні обчислення у кільці многочленів багатьох змінних.

Поле часток. У п.1.2 зазначалося, що будь-яку область цілісності (кільце без дільників нуля) D можна розширити до поля F. Мінімальне розширення $D \subset F$ називається полем часток над областю цілісності. Доведення цієї теореми є конструктивним: спочатку визначають поняття дробу $F = \{\frac{a}{b}, a,b \in D, b \neq 0\}$. На множині дробів визначають відношення еквівалентності $\frac{a_1}{b_1} \sim \frac{a_2}{b_2} \Leftrightarrow a_1b_2 = a_2b_1$. Клас еквівалентних дробів за означенням є елементом F. Арифметичні операції над F визначають конструктивно та доводять, що відношення $\frac{a_1}{b_1} \sim \frac{a_2}{b_2}$ є конгруенцією.

Нарешті, вкладення $D \to F$ визначають еквівалентністю $a \sim \frac{a}{1}$. Таким чином, у нашій термінології поле часток F є статичним розширенням області цілісності D, причому специфікації операцій точно слідують математичним конструкціям.

У прикладі 3.9 ці конструкції відтворені для побудування поля *Rat* як статичного розширення області *Int*. Крім *Rat*, полями часток є алгебри раціональних виразів однієї змінної та багатьох змінних, раціональних тригонометричних виразів, деякі інші алгебри. Діаграму відповідних абстракцій представлено на рис 3.9.

Обчислення у *QuotField* визначаються з точністю до арифметичних операції у *IntDomain*. Функції *Form, GCD* є віртуальними. Таким чином, для ефективної реалізації обчислень у цих алгебрах, *QuotField* достатньо реалізувати засобами C++. Зауважимо, що кільця *Int* та *Polynomial* є евклідовими: у цих кільцях визначені операції *div, mod* з використанням яких можна сформулювати алгоритм Евкліда обчислення *GCD*. Таким чином, якщо поле часток специфікувати над евклідовою областю, алгоритм Евкліда також

можна реалізувати засобами мови реалізації C++. Оскільки скорочення дробів виконується кожного разу при обчисленнях, це призводить до суттєвого скорочення часу обчислень.

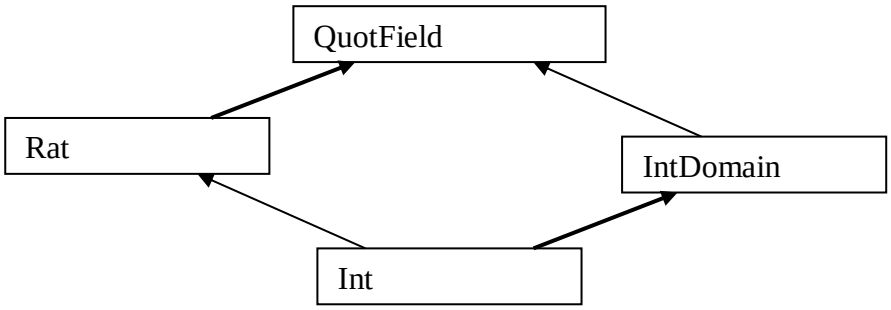


Рис. 3.9. Діаграма специфікацій полів часток

Детальніше з технікою програмування символьних обчислень засобами C++ можна ознайомитися в [59, 60].

3.7. Напівконструктивні означення алгебр та метод параметризації

В математиці, зокрема, в алгебрі, часто розглядають означення алгебр, які поєднують як конструктивні, так і абстрактні риси. Наприклад:

1. Кільце поліномів змінної x над полем F можна задати канонічною формою поліному $p(x) = a_0 x^n + a_1 x^{n-1} + ... + a_n$ та формулами операцій кільця, які оперують з коефіцієнтами $a_0, a_1, ..., a_n$. Конструктивними рисами у цьому означенні є форма представлення поліному та формули кільцевих операцій над поліномами. Абстракцією є визначення поля F. Це – довільне, тобто абстрактне поле.

2. Прямий добуток $G \times H$ груп G, H, пряма сума векторних просторів, алгебра $n \times n$ матриць.

Ці приклади можна продовжувати. Зокрема, приклади специфікацій простору лінійних комбінацій та поля часток, є також прикладами напівконструктивних означень. Дійсно, простір *AlinComb* є множиною лінійних комбінацій (конструктивна риса), а мономи визначені абстрактно (аксіомами векторного простору). Поле часток визначено як множина дробів (як конструкція), а чисельники та знаменники – абстрактно (аксіомами області цілісності).

У цих випадках природнім є використання підходу, який у ООП називають інстанціюванням, або параметризацією [60-63]. У цій термінології напівконструктивно визначені алгебри є шаблонами (templetes) або параметризованими класами. Шаблони задають конкретні класи, якщо замість параметрів підставити їх конкретні значення. Отже, з цього погляду звичайне математичне позначення кільця поліномів однієї змінної *F[x]* над полем F є шаблоном з параметрами F, x. Для специфікацій *F[x]* пропонується використовувати нотацію

Polynom(Field Coef, Const Atom Variable).

Polynom – ім'я алгебри-шаблону, *Field Coef* – відповідно тип та ім'я першого параметру, *Const Atom Variable* – відповідно кваліфікатор, тип та ім'я другого параметру. Надамо необхідні пояснення: пропонується надавати параметрам *типи*, які є іменами абстрактних алгебр та локальні *імена,* які використовують для специфікацій цієї алгебри і за якими можна відрізняти параметри, якщо вони мають один і той же тип. Кваліфікатори *Const* та *Var* розрізняють конкретний елемент алгебри від алгебри як елемента. Якщо кваліфікатор відсутній, за замовченням це *Var*. У прикладі, що розглядається, параметром є змінна – елемент алгебри *Atom.* Для специфікації кільця поліномів багатьох змінних, змінні якого можуть мати будь-які імена, ми використали б нотацію

Polynom(Field Coef, Var Atom Variable).

Звичайно, допускаються і інші кваліфікатори, якщо це потрібно для уточнення способів використання параметрів. Наприклад, можна у специфікаціях використовувати параметри-функції, визначені зовні специфікацій алгебр-параметрів. Кваліфікатором є службове слово *Function*.

Підстановка замість параметрів конкретних значень специфікує конструктивну алгебру, яка називається реалізацією. Наприклад: *RatPolX = Polynom(Rat, x)* – кільце поліномів змінної *x* над полем *Rat* з ім'ям *RatPolX*. Допускаються також часткові реалізації, які є по суті напівконструктивними алгебрами. Наприклад:

RatPol = Polynom(Rat, ...), PolX = Polynom(..., u).

Специфікації

VectorSpace(Field Coef), ALinComb(VectorSpaceAMonom, Atom Variable)

задають абстрактний векторний простір *ALinComb* як напівконструктивну алгебру (шаблон). Зауважимо, що параметр *Coef* у цьому означенні визначається у параметрі *AMonom*.

3.8. Використання методу морфізмів для специфікації алгебраїчних обчислень

Як зазначалося, метод морфізмів є одним з основних методів специфікацій багатосортних алгебр. У прикладі 3.5 відзначено одну з важливих ситуацій, в якій для специфікацій багатосортної алгебри застосовуються ізоморфізми – специфікації конвертацій представлень елементів алгебр. Насправді необхідність у перетвореннях даних, які по суті є ізоморфізмами алгебр, виникає на етапі введення даних, існує протягом виконання обчислень та завершується етапом відображення даних.

1. На етапі введення даних або з клавіатури, або з інших джерел математичні формули мають бути представлені у стандартних формах (*MathML*, *OpenMath* тощо). Система алгебраїчного програмування (наприклад, APS) пропонує користувачу свій власний формат. Таким чином, має бути реалізованим перетворення $MathML \Leftrightarrow APS$, яке по суті є ізоморфним перетворенням математичних виразів. Вимога сумісності системи APS з іншими математичними пакетами (Maple, Mathematica) означає необхідність програмування конверторів $MathML \Leftrightarrow OpenMath$.

2. У зв'язку з вимогами ефективності алгоритмів алгебраїчних обчислень треба реалізувати перетворення представлень елементів багатосортної алгебри. Перелічимо для ілюстрації основні канонічні форми тільки поліномів однієї змінної:

- у вигляді суми мономів, упорядкованих за спаданням степенів – для введення-виведення;
- у рекурсивному вигляді (схема Горнера) – для ефективного алгоритму обчислення значення многочлену;
- у вигляді списку "коефіцієнт-степінь" – для ефективного використання пам'яті та алгоритмів лінійних операцій додавання та множення на число;
- у вигляді вектора коефіцієнтів – для ефективних алгоритмів множення;
- у вигляді вектора коефіцієнтів «молодша половина – старша половина» – для алгоритму множення Карацуби;
- у вигляді вектора коефіцієнтів «мономи парних степенів – мономи непарних степенів" – для алгоритму дискретного перетворення Фур'є.

Зауважимо, що поряд з ізоморфними відображеннями алгебр, деякі важливі алгоритми потребують також гомоморфних відображень. Класичним прикладом є алгоритм Берлекемпа факторизації поліномів, першим етапом якого є гомоморфне відображення поліному з цілими коефіцієнтами у поліном з коефіцієнтами зі скінченного поля *GF(p)*.

Оскільки ізоморфізми визначаються як відображення носіїв подібним алгебр, їх специфікація здійснюється за діаграмою

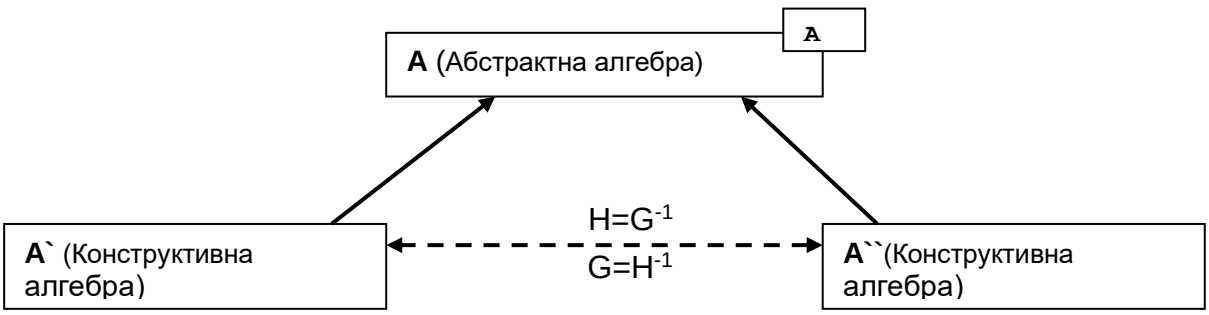


Рис. 3.10. Діаграма специфікацій ізоморфізмів алгебр

Продовжимо розгляд прикладу 5 до означення 3.15, у якому визначається поле *Rad* квадратних радикалів.

Приклад 3.17. Обчислення у полі *Rad* квадратних радикалів.

Нашою метою є розв'язання задачі спрощення раціональних числових виразів, які містять квадратні корені з раціональних чисел. Наведемо приклад такого виразу:

$$A = \frac{(2\sqrt{20} + 3\sqrt{12})^2 + \sqrt{\frac{4}{15}}}{(5+\sqrt{6})(2-3\sqrt{\frac{5}{12}})}$$

Результатом такого спрощення є цілий числовий вираз, який можна представити у вигляді лінійної комбінації коренів $\sqrt{2}, \sqrt{3}, \sqrt{5}$ та їх добутків з раціональними коефіцієнтами

$$A = r_0 + r_1\sqrt{2} + r_2\sqrt{3} + r_3\sqrt{5} + r_4\sqrt{6} + r_5\sqrt{10} + r_6\sqrt{15} + r_7\sqrt{30}\ .$$

Отже, у цій задачі маємо справу з полем *Rad1*, елементи якого мають вигляд $B = r_0 + \sum_i r_i\sqrt{q_i}$, де r_i – раціональні числа, q_i – натуральні числа, вільні від квадратів, причому $q_1 < q_2 < ... < q_i < ....$ Діаграма специфікацій алгебр для нашого прикладу має вигляд:

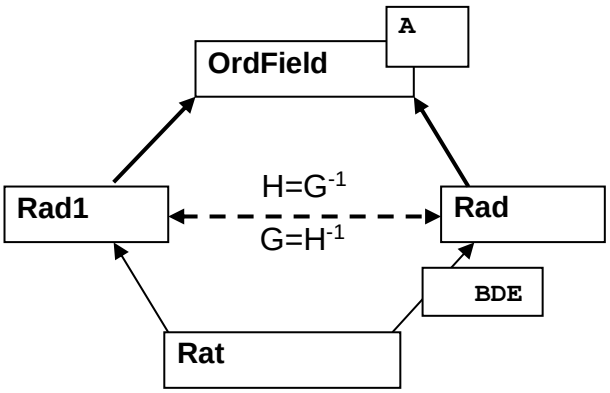


Рис. 3.11. Діаграма специфікацій алгебр прикладу 3.17.

Оскільки *Rad1* є впорядкованим полем, ми маємо визначити абстрактне впорядковане поле *OrdField*, яке наслідує *Field, LinOrd* та містить аксіоми, які пов'язують операції поля та відношення порядку:

```
Sort OrdField::Field, LinOrd;
    Axioms
    a < b -> a+c < b+c,
    (c > 0)&(a < b) -> a*c < b*c,
    a<>0 -> a*a > 0;
    …
```

Поле *Rad1* є розширенням *Rat*. Поле *Rad* є бінарним динамічним розширенням *Rat*. Один з можливих ефективних підходів до реалізації алгоритмів спрощення виразів у полі *Rad1* полягає у реалізації ізоморфізму $H: Rad1 \to Rad$, обчисленні значення виразу A у полі *Rad,* і реалізації зворотного гомоморфізму $G: Rad \to Rad1$.

Оскільки ізоморфізм H лишає нерухомими раціональні числа, достатньо реалізувати його для коренів квадратних з раціональних чисел: $H(\sqrt{\frac{a}{b}} = \frac{1}{b} H(\sqrt{ab}) = \frac{p}{b}\sqrt{q}$, де числа p, q є результатом застосування алгоритму звільнення від квадратів до підкорінного виразу ab.

Позначимо через $rad(a,b,p)$ елемент поля *Rad*, а через *SqrFree* – функцію звільнення від квадратів натурального числа. *SqrFree: Nat -> Rad*. Функція *SqrFree* задовольняє умові

$$SqrFree(x) = rad(0, y, p) \to x = y^2 p,$$

де p не містить множників – повних квадратів. Таким чином, $H(\sqrt{\frac{a}{b}}) = \frac{1}{b} * SqrFree(ab)$. У нашому прикладі

$$H(\sqrt{20}) = rad(0,2,5), H(\sqrt{12}) = rad(0,2,3), H(\sqrt{\frac{4}{15}}) = rad(0, \frac{2}{15}, 15),$$
$$H(\sqrt{6}) = rad(0,1,6), H(\sqrt{\frac{5}{12}}) = rad(0, \frac{2}{12}, 15)$$

.

2-ий етап обчислень – обчислення у полі *Rad*. На третьому етапі треба застосувати зворотній ізоморфізм $G = H^{-1}$. *Rad* є бінарним динамічним розширенням *Rat*. Тому в традиційних математичних позначеннях $A = a + b\sqrt{p}$, де a, b взагалі кажучи, мають той же вид. Для упорядкування радикалів вводимо функцію доступу *Index:* $Index(rad(a,b,p) = p$ з контекстними умовами

$$Index(A) > \max(Index(a), Index(b)),$$
$$GCD(Index(A), Index(a)) = 1,$$
$$GCD(Index(A), Index(b)) = 1$$

Це означає, що

$$a = a_1 + b_1\sqrt{p_1},\ b = a_2 + b_2\sqrt{p_2},$$

$$p_1 < p, p_2 < p, \gcd(p, p_1) = 1, \gcd(p, p_2) = 1. \qquad (3.23)$$

Отже, ізоморфізм G задається співвідношенням $G(rad(a,b,p)) = a + b\sqrt{p}$, розкриттям дужок $b\sqrt{p} = a_2\sqrt{p} + b_2\sqrt{p_2}\sqrt{p}$, множенням радикалів $\sqrt{p_2}\sqrt{p} = \sqrt{p_2 p}$ та спрощеннями, які визначені для нуля та одиниці поля. Отже, специфікації ізоморфного відображення H, якому (за традицією) надано ім'я *Rad1ToRad* мають вид:

```
Sort Rad1::OrdField;
    Parameter Rat;
    Signature
     RadToRad1(1): Rad -> Rad1;
     Sqrt(1): Rad -> Rad1;
    Operations
     RadToRad1:     rad(a,b,p) = a + b*Sqrt(p);
     Sqrt:
    (a < 0) -> Sqrt(a))= Exeption('Square Root from negative
number'),
    Sqrt(0) = 0,
    Sqrt(p)*Sqrt(q) = Sqrt(p*q);
    Add:  a+b = RadToRad1(Rad1ToRad(a)+Rad1ToRad(b)),
    Mult: a*b = RadToRad1(Rad1ToRad(a)*Rad1ToRad(b)),
    Div:  a/b = RadToRad1(Rad1ToRad(a)/Rad1ToRad(b)),
    …

    Sort Rad::OrgField;
    Parameter Rat;
    Signature
     Rad1ToRad(1): Rad1 -> Rad;
     SqrFree: Nat -> Rad;
    Operation
    Rad1ToRad:
    (a//b>0)->Rad1ToRad(Sqrt(a//b))= rad(0,1//b*SqrFree(a*b)),
      //частковий випадок натуральне число під знаком радикалу
    (a>0) -> Rad1ToRad(Sqrt(a)=rad(0,SqrFree(a)),
    …
```

Наведемо специфікації операцій додавання, множення та ділення у полі Rad.

```
    Add:
    rad(a,b,p)+rad(c,d,p)=rad(a+c,b+d,p),
    (p>q)&(gcd(p,q)=1)->
    rad(a,b,p)+rad(c,d,q)=rad(a+rad(c,d,q),b,p),
    (p<q)&(gcd(p,q)=1)->
    rad(a,b,p)+rad(c,d,q)=rad(rad(a,b,p)+c,d,q),
    gcd(p,q)<> 1 ->
    rad(a,b,p)+rad(c,d,q)=          a+rad(0,b,(p          div
    gcd(p,q))*rad(0,1,gcd(p,q))+
       (c+rad(0,d,(q div gcd(p,q))*rad(0,1,gcd(p,q));

    Mult:
    rad(a,b,p)*rad(c,d,p)=rad(a*c+b*d*p,a*d+d*c,p),
```

```
    (p>q)&(gcd(p,q)=1)->
      rad(a,b,p)*rad(c,d,q)=rad(a*rad(c,d,q),b*rad(c,d,q),p),
    (p<q)&(gcd(p,q)=1)->
      rad(a,b,p)*rad(c,d,q)=rad(rad(a,b,p)*c,rad(a,b,p)*d,q),
    gcd(p,q)<> 1 -> rad(a,b,p)*rad(c,d,q)=
    (a+rad(0,b,(p div gcd(p,q))*rad(0,1,gcd(p,q)) *(c+rad(0,d,(q
div gcd (p,q))* rad(0,1,gcd(p,q));
    Div:
    rad(a,b,p)/rad(c,d,p)=
         rad(a,b,p)*rad(c/(c^2-d^2*q), -d/(c^2-d^2*q),q);
```

Надамо необхідні пояснення до правил операції додавання та множення. Контекстні умови конструктора *rad* (3.23) потребують, зокрема, взаємної простоти радикалів, які визначають індекси в бінарному алгебраїчному розширенні. Тому при додаванні або множенні чисел $a+b\sqrt{p}$, $c+d\sqrt{q}$, перш, ніж перевіряти нерівності $p<q, q<p$ слід забезпечити взаємну простору індексів: $\gcd(p,q)=1$. Позначимо

$$d=\gcd(p,q),\ p_1=p\,divd,\ q_1=q\,divd.$$

Тоді

$$a+b\sqrt{p}=a+b\sqrt{p_1}*\sqrt{d},\ c+d\sqrt{q}=c+d\sqrt{q_1}*\sqrt{d} \qquad (3.24)$$

В цій формулі радикали задовольняють умові взаємної простоти. Тому, якщо $\gcd(p,q)\neq 1$, ми виконуємо попередньо перетворення (3.24) , а потім – операцію додавання або множення.

Оскільки поле *Rad* є динамічним розширенням *Rat*, специфікації операцій мають бути доповнені правилами, один з аргументів яких є елементом *Rat*. Для того, щоб уникнути цих доповнень, можна представити раціональне число r у вигляді $r=Rad(r,0,1)$ ($r=r+0*\sqrt{1}$). Тоді, як легко перевірити, при виконанні операції у випадку, коли один з її аргументів – раціональне число, виконається одне з перших трьох правил, оскільки $(\gcd(p,1)=1)\,\&\,(\gcd(1,q)=1$. Нарешті, специфікації *Rad* мають визначати відношення строгого порядку $rad(a,b,p)<rad(c,d,q)$, яке є продовженням цього відношення на *Rat*.

```
(rad(a,b,p)<rad(c,d,p))=(rad(a-c,b-d,p)<0),
(p<q)->(rad(a,b,p)<rad(c,d,q))=rad(a-c+rad(0,b,p),-d,q)<0,
(p>q)->(rad(a,b,p)<rad(c,d,q))=rad(a-c-rad(0,d,q), b,q <0,
rad(a,b,p)<0=
       (b<0)&((a<=0)|(a>0)&(b^2*p>a^2))|(b>0)&(a<0)&(b^2*p<a^2));
```

Ці обчислення по суті зводять нерівності виду $A<B$ до нерівностей $A-B<0$ та відповідають структурі *Rad* як бінарного динамічного розширення *Rat*. Тому їх можна реалізувати за допомогою функції

```
Neg(1):Rad -> Bool, Neg(a) = a < 0 .
(rad(a,b,p)< rad(c,d,p)) = Neg(rad(a-c,b-d,p)),
(p<q)->(rad(a,b,p)<rad(c,d,q)=Neg(rad((a-c+rad(0,b,p),-d,q)),
(p>q)->(rad(a,b,p)<rad(c,d,q))=Neg(rad((a-c-rad(0,d,q), b,q)),
Neg(a,b,p)= Neg(b)&(Neg(a)|(a=0)|Neg(-a))&Neg(a^2-b^2*p))
      |Neg(-b)&Neg(a)&Neg(b^2*p-a^2));
```

3.9. Висновки: парадигма алгебраїчних обчислень

Практика реалізації обчислень у багатосортних алгебрах показала, що запропонований підхід є по суті універсальним. Цей підхід було розглянуто детально, починаючи з означень багатосортної алгебри і закінчуючи застосуванням методів розширень, наслідування та морфізмів. Численні приклади всебічно демонструють його. Заключний приклад 3.17, ілюстрований діаграмою (рис 3.10), показує роль і місце методів розширень, наслідувань та морфізмів як основних методів алгебраїчних обчислень. У наступному розділі цей підхід буде застосований для реалізації тригонометричних обчислень та обчислень в алгебрах числових множин. Таким чином, наш підхід можна вважати *парадигмою алгебраїчних обчислень*.

Глава 4. Методи комп'ютерної алгебри розв'язання тригонометричних задач

4.1. Основи комп'ютерної тригонометрії

Тригонометрія як математична дисципліна займає особливе місце. Тригонометрію вивчають як окремий розділ шкільної алгебри і широко використовують в інших математичних, фізичних та інженерних дисциплінах. Тому і математичні системи мають підтримувати тригонометричні обчислення. Цю главу присвячено методам розв'язання тригонометричних задач шкільного курсу алгебри.

4.1.1. Цілі тригонометричні вирази. Тригонометричні поліноми

Абстрактна алгебра тригонометричних поліномів. Тригонометричними функціями будемо називати вирази виду $Sin(X)$, $Cos(Y)$, де $X, Y-$ лінійні комбінації змінних та константи *Pi* з раціональними коефіцієнтами. Тригонометричним одночленом будемо називати добуток декількох тригонометричних функцій. Цілим тригонометричним виразом (тригонометричним поліномом, Т-поліномом) будемо називати поліном від тригонометричних одночленів.

Точне означення Т-полінома задається специфікацією відповідної алгебри як члена ієрархії багатосортної алгебри.

```
Sort TArg::AlinComb;
    Parameters
     TArgCoef = ArgCoef;
    Signature
     Const Pi;
    …
```

Сорт *TArg* на абстрактному рівні визначає лінійний простір аргументів Т-поліномів. Суть визначення полягає у тому, що

- над аргументами Т-поліномів здійснюються лише лінійні операції;
- в аргументах Т-поліномів може використовуватися константа π.

```
Sort TCoef::Field;
```

…

Сорт `TCoef` визначає поле коефіцієнтів Т-поліномів.

```
Sort TPolynom::LinAlg;
    Parameters
    TArgCoef, TCoef;
    Signature
     Sin(1):TArg -> TPolynom;
     Cos(1):TArg -> TPolynom;
```

```
Axioms
Sin(0) = 0, Sin(Pi) = 0, Cos(0) = -1, Cos(Pi) = 1,
Sin(-x) = -Sin(x), Cos(-x)= Cos(x),
Sin(x+y) = Sin(x)*Cos(y) + Cos(x)*Sin(y),
Cos(x+y) = Cos(x)*Cos(y) - Sin(x)*Sin(y);

Isomorphism
FiPolToTPol:FiPolynom -> Tpolynom:
FiPolToTPol(Fi(a,b,x) = a*Cos(x)+b*Sin(x);

Operations
Sin: Sin(0) = 0, Sin(Pi)= 0;
Cos: Cos(0) = 1, Cos(Pi)= -1;
…
```

Сорт *TPolynom* на абстрактному рівні визначає алгебру Т-поліномів як лінійну алгебру. Таким чином, на множині Т-поліномів визначені операції додавання, віднімання, множення на коефіцієнт, множення та піднесення до натурального степеня. Тригонометричні функції визначаються як унарні операції з аргументами з лінійного простору *TArg* та значеннями у *Tpolynom;*

$$Sin, Cos: TArg \to Tpolynom.$$

Аксіоми сорту *TPolynom* визначають основні властивості тригонометричних функцій. На абстрактному рівні визначені константи $0, 1, Pi$. Тому правила обчислення значень тригонометричних функцій містять спрощення, що оперують з цими константами.

4.1.2. Конструктивна реалізація алгебри тригонометричних поліномів

```
Sort PiMonom = LinMonom(Rat, Const Pi);
```

Сорт *PiMonom* є параметричною реалізацією сорту *LinMonom.* Тому елементи сорту мають вид $a\$Pi, a \in Rat, Pi$ – константа.

```
Sort VarMonom = LinSpace(LinMonom(Rat, Variable));
```

Сорт *VarMonom* є параметричною реалізацією сорту *ALinSpace*. Елементи сорту є лінійними комбінаціями виду

$$a_1\$x_1 + + a_2\$x_2 + + \ldots + + a_n\$x_n,\ a_i \in Rat, x_i \in Variable$$

```
Sort FiArg::VectorSpace;
Parameters

VarMonom, PiMonom;

Constructor{
FiArg T = VarMonom X ++ PiMonom A;
X ++ 0 = x, 0 ++ A = A;    // Функції вкладення
VarArg(X ++ A) = X;        // Функції доступу
```

```
PiArg(X ++ A) = A;
LeadCoef(X++A)=LeadCoef(X);
LeadCoef(A)=Cf(X);
Ind(X++A)=Ind(X);
Ind(A)=Cf(A);
}
```

Operations

```
Add:  (x++a)+(y++b)= (x+y)++(a+b);
Sub:  (x++a)-(y++b)= (x-y)++(a-b);
CMult: c*(x++a) = c*x ++ c*a;
…
```

Сорт *FiArg* визначає тригонометричний аргумент як суму лінійної комбінації змінних та вільного члену. Таким чином, *FiArg* є статичним розширенням сортів *PiMonom* та *VarMonom* з функціями вкладення за першою та другою координатами, які по суті є функціями проектування. Функції доступу виділяють окремо лінійну комбінацію змінних, вільний член та старший коефіцієнт аргументу як старший коефіцієнт лінійної комбінації. Функція доступу *Ind* визначає індекс тригонометричного аргументу.

Операції векторного простору визначені покоординатно. Таким чином, алгебра *FiArg* є по суті прямою сумою векторних просторів *VarMonom* та *PiMonom*.

```
Sort FiMonom::AMonom;
```

Parameters Field FiCoef, FiArg FiArg;

Constructor{

```
FiMonom T = Fi(FiCoef a, FiCoef b, FiArg X);
/* Конструктор сорту */
CosCoef(T) = a,        // Функції доступу
SinCoef(T) = b,
Arg(T) = X;
Fi(A, B, 0) = a,       // Функція вкладення
Fi(A, B, Pi) = b;
LeadCoef(X) > 0,       // Контекстні умови
Cf(PiArg(x)) >= 0,
Cf(PiArg(x))< 1//2;
};
```

Signature

// Ізоморфізм TPolynom -> FiMonom

```
TPolToFiMon(1): TPolynom -> FiMonom;
```

Isomorphism

```
TpolToFiMon:
     Sin(X)= Fi(0,1,X),
     Cos(X)= Fi(1,0,X);
```

Operations

Fi: // Інтерпретатор конструктора

```
     LeadCoef(X)<0 -> Fi(a,b,X)= Fi(a,-b,-X),
```

// Формули зведення зводять додатковий кут у інтервал [0, Pi/2)

```
     Cf(PiArg(X))>= 2 ->  Fi(a,b,X) = Fi(a,b, X-2*Pi),
```

```
        Cf(PiArg(X))<= -2 -> Fi(a,b,X) = Fi(a,b, X+2*Pi),
        Cf(PiArg(X))>= 1 ->  Fi(a,b,X) = Fi(-a,-b,X-Pi),
        Cf(PiArg(X))< 0 ->   Fi(a,b,X) = Fi(-a,-b,X+Pi),
        Cf(PiArg(X))>= 1//2 -> Fi(a,b,X) = Fi(b,-a,X-1//2*Pi);
    Add:   Fi(a,b,X)+ Fi(c,d,X)= Fi(a+c, b+d, X);
    Sub:   Fi(a,b,X)- Fi(c,d,X)= Fi(a-c, b-d, X);
    CMult: c*Fi(a,b,X)= Fi(c*a, c*b, X);
    …
```

Сорт *FiMonom* визначає тригонометричний моном формулою

$$Fi(a, b, X) = a*Sin(X) + b*Cos(X). \qquad (4.1)$$

Таким чином,

$$Sin(X) = Fi(0, 1, X), \quad Cos(X) = Fi(1, 0, X).$$

Для того, щоб представлення (4.1) було канонічним, необхідно, щоб старший коефіцієнт тригонометричного аргументу був додатним та додатковий кут знаходився у інтервалі $[0, Pi/2)$. Крім того, мають бути задоволені деякі інші додаткові умови, формулювання яких залежить від поля коефіцієнтів *FiCoef*. Тому відмітка *Fi* конструктора сорту *FiMonom* має бути інтерпретованою. Її інтерпретатор використовує властивості парності/непарності цілих тригонометричних функцій та відомі формули зведення.

Основна ідея обчислень у кільці Т-поліномів полягає у ізоморфному переході від тригонометричного виразу в сигнатурі *TPolynom* до виразу в сигнатурі *FiPolynom*, побудові канонічної форми цього виразу як елемента *FiPolynom*, та оберненому ізоморфному переході у сигнатуру *TPolynom*. Отже, до реалізації тригонометричних обчислень застосовується парадигма алгебраїчних обчислень:

```
TCan(P) = FiPolToTPol(FiCan(TPolToFiPol(P))).
```

На завершення пункту розглянемо специфікації сорту *FiPolynom*.

```
FiPolynom=LinSpace(FiMonom(FiCoef,Variable))::LinAlg,LinOrd;

Signature
 FiMult(2),()*():FiMonom*FiMonom->FiPolynom;

Operations
 FiMult:Fi(a,b,X)*Fi(c,d,Y)=       Fi((a*c-b*d)/2,(b*c+a*d)/2,X+Y)
+Fi((a*c+b*d)/2,(b*c-a*d)/2,X-Y);
  Mult: (a++A)*(b++B)= (a*b + (a*B + A*b))+A*B;
  Less: (a++A)<(b++B) = (a<b)|(a==b)&(A<B);
…
```

4.1.3 Канонічні форми раціональних тригонометричних виразів

Нехай F, G – Т-поліноми однієї змінної x. Раціональний вираз $Y = F/G$ будемо називати TR – виразом. Привівши F, G до канонічної форми з використанням функції $Fi(a,b,x) = aCos(x) + bSin(x)$, отримаємо:

$$H(x)=\frac{Fi(a_k,b_k,kx)+...+Fi(a_1,b_1,x)+a_0}{Fi(c_m,d_m,mx)+...+Fi(c_1,d_1,x)+c_0}.$$

Основна ідея побудови канонічної форми $H(x)$ – обчислення найбільшого спільного дільника чисельника й знаменника цього представлення й скорочення на нього – в точності повторює алгоритм спрощення раціонального алгебраїчного виразу однієї змінної, в якому $Fi(a_j,b_j,jx)$, $Fi(c_j,d_j,jx)$ відіграють роль мономів.

Однак, легко показати, що Т-поліноми розкладаються в добуток тригонометричних поліноміальних співмножників неоднозначно. Наприклад,

$$\frac{1}{2}-\frac{1}{2}Cos(2x)=Sin(x)*Sin(x)=(1-Cos(x))(1+Cos(x)).$$

Переходячи до функції Fi, отримаємо:

$$Fi(\frac{1}{2},0,2x)+\frac{1}{2}=Fi(0,-1,x)\cdot Fi(0,1,x)=[Fi(1,0,x)+1][Fi(1,0,x)-1]$$

Таким чином, має місце рівність TR – виразів

$$\frac{\sin(x)}{1+\cos(x)}=\frac{1-\cos(x)}{\sin(x)}, \qquad (4.2)$$

чисельники і знаменники яких є незвідними у кільці Т-поліномів. Істотним є також вибір поля коефіцієнтів: допустивши комплексні коефіцієнти, рівність (4.2) можна продовжити:

$$\frac{\sin(x)}{1+\cos(x)}=\frac{1-\cos(x)}{\sin(x)}=\frac{i-i\cos(x)+\sin(x)}{1+\cos(x)+i\sin(x)}$$

Для усунення цих неоднозначностей зафіксуємо, по-перше, поле коефіцієнтів *Coef* поля TR-виразів, і, по-друге, визначимо структуру цього розширення над полем коефіцієнтів. Нехай $Coef = Q$ – поле раціональних чисел або деяке його дійсне алгебраїчне розширення (наприклад, поле ***Rad***). Тим самим ми виключили можливість використання комплексних коефіцієнтів. Тоді полем TR–виразів є поле $Coef(Fi(a,b,kx)), a,b\in Coef, k\in N$. Оберемо й зафіксуємо послідовність розширень полів

$$Coef\subset Coef(\cos(x))\subset Coef(\cos(x))(\sin(x)),$$

приєднавши до *Coef* послідовно елементи $\cos(x)$, $\sin(x)$. Для стислості введемо позначення $Coef=F_0$, $F_0(\cos(x))=F_1$, $F_1(\sin(x))=F_2$. Розширення $F_0\subset F_1$ є трансцендентним, а розширення $F_1\subset F_2$ – алгебраїчним. Тому елемент H

поля F_2 можна представити у вигляді $H = f(x)/g(x)$, де $f, g \in F_1[\sin(x)]$. Поліноми f, g записані у вигляді суми мономів від ступенів $\cos^j(x)$, формулами пониження степенів можуть бути переписані в поліноми від $\cos(jx)$.

$$f(x) = a_0 Cos(kx) + ... + a_{k-1} Cos(x) + a_k, \; g(x) = b_0 Cos(mx) + ... + b_{m-1} Cos(x) + b_m$$

,

Тому елемент $H \in F_1$ можна представити у вигляді:

$$H = \frac{f(x)}{g(x)} = \frac{a_0 Cos(kx) + ... + a_{k-1} Cos(x) + a_k}{b_0 Cos(mx) + ... + b_{m-1} Cos(x) + b_m} \tag{4.3}$$

При цьому T-поліноми f, g взаємно прості: $\gcd(f, g) \in F_0$. Це представлення, однак, не є однозначним: для канонічності знаменник дробу (4.3) потрібно нормалізувати, розділивши чисельник і знаменник на $lc(g) = b_0$:

$$H = \frac{f(x)}{g(x)} = \frac{a_0' Cos(kx) + ... + a_{k-1}' Cos(x) + a_k'}{Cos(mx) + ... + b_{m-1}' Cos(x) + b_m'}, \quad a_j' = \frac{a_j}{b_0}, b_l' = \frac{b_l}{b_0}.$$

Таке представлення є канонічним. Оскільки $\cos^2(x) + \sin^2(x) = 1$, ступінь розширення $F_1 \subset F_2$ дорівнює 2. Тому елемент поля F_2 можна представити у виді лінійного двочлена:

$$H \in F_2 \Rightarrow H = G_0 \sin(x) + G_1, G_0, G_1 \in F_1.$$

Таким чином, канонічною формою TR-виразу є його представлення у вигляді

$$\begin{aligned} &H = \frac{f_0(x)}{g_0(x)} \sin(x) + \frac{f_1(x)}{g_1(x)}, \\ &f_0, g_0, f_1, g_1 \in F_1 = F_0(\cos(x)), . \\ &\gcd(f_0, g_0) = 1, \gcd(f_1, g_1) = 1, \\ &lc(g_0) = 1, lc(g_1) = 1. \end{aligned} \tag{4.4}$$

Для розв'язання задачі залишилося вказати алгоритм обернення цілого тригонометричного виразу, тобто алгоритм побудови канонічної форми виразу виду

$$H(x) = \frac{1}{\sin(x) f(x) + g(x)}, \quad f(x), g(x) \in F_1 \tag{4.5}$$

Покладемо $A = f(x), B = g(x), u = Cos(x), v = Sin(x)$. Тоді

$$\frac{1}{vA+B} = vP+Q,\ P,Q \in F_1 \tag{4.6}$$

Рівняння (4.6) відносно P,Q розв'яжемо у полі $F_2 = F_0(u,v)/\{v^2+u^2=1\}$, помноживши обидві частини на сполучену величину $-vA+B$:

$$\frac{B-vA}{B^2-v^2A^2} = \frac{B-vA}{B^2-(1-u^2)A^2} = \frac{B-vA}{B^2-A^2+u^2A^2} = \frac{B}{B^2-A^2+u^2A^2} + v\frac{-A}{B^2-A^2+u^2A^2}.$$

Звідси

$$P = \frac{B}{B^2-A^2+u^2A^2},\ Q = \frac{-A}{B^2-A^2+u^2A^2}. \tag{4.7}$$

Формули (4.7) представляють правило обернення виразу (4.5). Обговоримо на закінчення переваги й недоліки деяких різних канонічних форм TR-виразів.

Нехай $f(x)$ – T-поліном, $m = \deg f$. Використовуючи показову функцію уявного аргументу ix, виразимо $Cos(kx), Sin(kx)$ через e^{ikx}:

$$Cos(kx) = \frac{e^{ikx}+e^{-ikx}}{2},\ \ Sin(kx) = \frac{e^{ikx}-e^{-ikx}}{2i}$$

Поклавши $e^{ikx} = y$, отримаємо:

$$aCos(kx) + bSin(kx) = \frac{a-ib}{2}y^k + \frac{a+ib}{2}y^{-k} \tag{4.8}$$

звідки випливає, що $f(x)$ можна представити у вигляді $f(x) = F(y)/y^m$, де $F(y)$ – поліном ступеня $2m$ з комплексними коефіцієнтами, коефіцієнти якого, рівновіддалені від кінців, сполучені: $A_j = \bar{B}_{2m-j},\ j = 0,...,2m$. Це ізоморфне перетворення можна використовувати для ефективної реалізації обчислень у полі тригонометричних виразів (наприклад, з використанням стандартних бібліотек поліноміальних обчислень). Пряме й зворотне ізоморфні перетворення (4.8) виконуються за лінійний час. Недолік полягає у необхідності обчислень з комплексними числами.

Канонічна форма (4.4) використовує обчислення з дійсними числами. Якщо в цій формі використовувати представлення f_0, f_1, g_0, g_1 у вигляді алгебраїчних поліномів (тобто здійснити ізоморфне перетворення $y = Cos(x)$), також можна використовувати ефективні реалізації обчислень у кільці поліномів однієї змінної. Однак, для деяких шкільних тригонометричних задач таке представлення може бути неефективним. Справді, розглянемо задачу розв'язання рівняння виду $Sin(100x) = Sin(x)$, або $Sin(100x) - Sin(x) = 0$. Канонічною формою лівої частини рівняння є поліном 100-го ступеня, у якому всі коефіцієнти відмінні від нуля. Тому f_0, f_1, g_0, g_1 краще представляти у формі

(4.1). Отже, найкращого з усіх поглядів представлення тригонометричного виразу не існує. Вибір цього представлення багато у чому суб'єктивний.

4.1.4. Поле коефіцієнтів тригонометричних поліномів

Як зауважено у попередньому пункті, сорт *TCoef* визначає поле коефіцієнтів Т-поліномів. У найпростішому випадку це поле можна специфікувати як *Rat*. Однак, навіть у простих тригонометричних задачах шкільного типу коефіцієнтами можуть бути квадратні радикали. Наприклад: $\mathrm{Sin}(\pi/4) = \sqrt{2}/2, \mathrm{Sin}(\pi/3) = \sqrt{3}/2$. Тому використання поля *Rad* дозволяє розширити клас тригонометричних задач, які підтримуються програмною системою. Однак найбільш прийнятним є розширення поля *Rad* тригонометричними функціями від числових аргументів виду $k\pi/n, k, n \in N$.

Позначимо через *TCoef* поле $Rad(Sin(k\pi/n), Cos(k\pi/n), k, n \in N)$. Оскільки $Sin(\alpha) = Cos(\pi/2 - \alpha)$, $TCoef = Rad(Cos(k\pi/n, k, n \in N)$. Числа $\cos(k\pi/n), \sin(k\pi/n)$ є алгебраїчними. Таким чином, для числа $\cos(\pi/n)$ над полем Q існує мінімальний поліном, тобто поліном $P_c(x) \in Q[x]$ мінімального степеня такий, що $P_c(\cos(\pi/n)) = 0$. Перетвореннями пониження степеня $\cos^k(x) = c_0 \cos(kx) + ... + c_{k-1}\cos(x) + c_k$ полінома $P_c(x)$ можна поставити у відповідність Т-поліном, який по суті є мінімальним тригонометричним поліномом числа $\cos(\pi/n)$. Надамо відповідне означення:

Означення 4.1. Мінімальним Т-поліномом алгебраїчного числа $a = \cos(\pi/n), n \in N$ над полем Q називається Т-поліном мінімального степеня m виду

$$T(x) = \cos(mx) + c_1 \cos((m-1)x) + ... + c_{m-1}\cos(x) + c_m, c_j \in Q$$

такий, що $T(a) = 0$.

Наступна теорема встановлює співвідношення між мінімальним поліномом алгебраїчного числа $\varepsilon_n = \cos(2\pi/n) + i\sin(2\pi/n)$ та мінімальним тригонометричним поліномом його дійсної частини $r_n = \cos(\pi/2n)$.

Теорема 4.1. Нехай $e_n = \cos(2\pi/n) + i\sin(2\pi/n)$ – корінь степеня n з одиниці, $P_n(z) \in Q[z]$ – мінімальний поліном e_n над полем Q, $T_n(x)$ – мінімальний Т-поліном числа $re(e_n) = \cos(2\pi/n)$ над Q та $m = \deg(P_n)$. Тоді для будь-якого z, $|z| = 1$, має місце співвідношення

$$P_n(z) = z^{m/2} \cdot T_n(re(z)) \tag{4.9}$$

Доведення. Нехай $P_n(z) = z^m + c_1 z^{m-1} + ... + c_{m-1}z + c_o$. Неважко показати, що множина коренів $P_n(z)$ замкнута відносно операції обернення z^{-1}, тобто якщо ε-корінь $P_n(z)$, то ε^{-1} – також корінь $P_n(z)$, причому $\varepsilon^{-1} = \bar{\varepsilon}$. Тому для будь-якого i $c_i = c_{m-i}$, m – парне число. Нехай $m = 2l$. Тоді

$$\frac{P_n(z)}{z^l} = (z^l + z^{-l}) + c_1(z^{l-1} + z^{-l+1}) + ... + c_{l-1}(z + z^{-1}) + c_l \qquad (4.10)$$

Якщо $|z| = 1$, $z = \cos(x) + i\sin(x), 1/z = \bar{z} = \cos(x) - i\sin(x)$. Отже, права частина рівності (4.10) є Т-поліномом від $re(z) = \cos(x)$. Позначимо цей Т-поліном через $T_n(x)$. Представлення (4.10) отримане. Залишилося показати, що $T_n(x)$ – мінімальний Т-поліном $\cos(2\pi/n)$. Зауважимо для цього, що перетворення $z^k + z^{-k} = 2\cos(re(z))$ можна використати і у зверненому напрямку: для будь-якого характеристичного Т-полінома $T(x)$: $T(re(z)) = c_0((z^k + z^{-k})/2) + ... + c_{k-1}((z + z^{-1})/2) + c_k$. Помноживши обидві частини рівності на z^k, отримаємо характеристичний поліном числа $\varepsilon_n = \cos(2\pi/n) + i\sin(2\pi/n)$. Отже, якщо $T_n(x)$ – мінімальний, то і $P_n(z)$ – мінімальний. Теорему доведено.

Наслідок. Якщо $P_e(z) = z^m + c_1 z^{m-1} + ... + c_{m-1}z + 1$ – мінімальний поліном числа $\varepsilon_n = \cos(2\pi/n) + i\sin(2\pi/n)$, $m = 2l$, мінімальний Т-поліном числа $\cos(2\pi/n)$ має вид

$T(x) = 2(\cos(lx) + c_1\cos((l-1)x) + ... + c_{l-1}\cos(x)) + c_l$.

Таким чином, для значення $x = 2\pi/n$ має місце співвідношення

$$\cos(2l\pi/n) = -c_1\cos(2(l-1)\pi/n - ... - c_{l-1}\cos(2\pi/n) - c_l/2 \qquad (4.11)$$

Приклад 4.1. Побудова мінімального Т-полінома числа $\cos(\pi/5)$.

Розглянемо число $\varepsilon_{10} = \cos(2\pi/10) + i\sin(2\pi/10)$.

Побудуємо мінімальний поліном ε_{10}: $h(z) = z^{10} - 1$.

$z^{10} - 1 = (z+1)(z^4 - z^3 + z^2 - z + 1)(z^5 - 1)$.

Звідки $P_{10}(x) = x^4 - x^3 + x^2 - x + 1$, $\deg(P_{10}) = m = 4, l = 2.$

Отже, $T_{10}(x) = 2\cos(2x) - 2\cos(x) + 1$, $2\cos(2\pi/5) - 2\cos(\pi/5) + 1 = 0$

Формула (4.11) використовується при побудові канонічної форми числового цілого Т-полінома. Далі ми наводимо алгоритм побудови таких форм, тобто алгоритм обчислень у полі Tcoef.

Означення 4.2. Числовим Т-поліномом називається вираз виду

$$F(\cos(k_1\pi/n_1), \cos(k_2\pi/n_2), ..., \cos(k_m\pi/n_m)), F(x_1, ..., x_m) \in Q[x_1, ..., x_m] \qquad (4.12)$$

Означення 4.3. Канонічною формою (стандартним видом) виразу (4.12) називається лінійна комбінація виду

$$L(F) = \sum_{j=0}^{M} c_j \cos(j\pi/n),\ M < n/2.$$

Алгоритм побудови канонічної форми цілого числового Т-полінома представимо у вигляді системи правил переписування:

```
TrigNumCan = rs(c,d,   k, P)     // Rat c,d 0 < c,d <1/2
(
 sin(c$Pi)=cos((1//2 - c)$Pi),       // Sin → Cos
```

```
 c<0 -> cos(c$Pi)=cos((-c)$Pi),     // c in (0;1//2)
 c>1//2 → cos(c$Pi)=cos((1-c)$Pi,
 cos(c$Pi)*cos(d$Pi)=(1//2)$Cos(c+d)$Pi)+(1//2)$Cos(c-
)$Pi),
```

/* елімінація множень *.

```
    k>2 → P^k= P^(k div 2))^2*P^(k mod 2),   //Інтерпретатор
```

Pow

```
    cos(c$Pi)^2  =  (1//2)$Cos(2*c)$Pi)+1//2,   // Елімінація
```

степенів

```
    c>Deg(c) → cos(c$Pi)=TrigNumRed(c)        // Редукція
);
```

Ключовим у даній системі переписувань є останнє правило. Оскільки алгоритм TrigNumRed(c) побудови мінімального Т-полінома числа $\cos(2\pi/n)$ залежить тільки від n, тобто знаменника дробу c, ефективність його роботи за часом залежить від часу побудови полінома $T_n(x)$. Звичайно, кожного разу викликати алгоритм обчислення $T_n(x)$ не треба: можна попередньо побудувати систему переписуючих правил для усіх $n \le N_0$ для деякого натурального N_0 (скажімо, $N_0 = 36$). Тоді для всіх значень кутів виду $k\pi/n,\ n \le N_0$ обчислення будуть виконуватися швидко. Для інших значень n треба виконати обчислення, викладені у теоремі 4.1, побудувавши правило редукції, а потім доповнити систему переписувань TrigNumRed(c) цим правилом.

Формула (4.9) зводить алгоритм побудови мінімального Т-полінома числа $\cos(2\pi/n)$ до алгоритма побудови мінімального полінома кореня $\varepsilon_1 = \cos(2\pi/n) + i\sin(2\pi/n)$. Далі ми наводимо одну з ефективних версій цього теоретико-числового алгоритму.

Алгоритм побудови мінімального полінома первісного кореня з 1.

Первісним коренем ступеня n з 1 називається такий корень e_j, що для будь-якого $k < n$ $e_j^k - 1 \ne 0$. Корінь $e_j = \cos(2\pi j/n) + i\sin(2\pi j/n)$ є первісним тоді і тільки тоді, коли j взаємно просте з n: $\gcd(j,n) = 1$. Корінь e_1 є первісним. Кількість первісних коренів степеня n з 1 дорівнює кількості $\varphi(n)$ натуральних чисел, менших ніж n та взаємно простих з n. Функція $\varphi(n)$ називається функцією Ейлера.

Лема 4.1. Нехай $M = \{\varepsilon_1, \varepsilon_2, ..., \varepsilon_{\varphi(m)}\}$, $N = \{\delta_1, \delta_2, ..., \delta_{\varphi(n)}\}$ – множини первісних коренів з 1 ступеня m й n відповідно. Визначимо множину $T = \{\varepsilon_i \cdot \delta_j : \varepsilon_i \in M,\ \delta_j \in N\}$. Якщо m, n – натуральні взаємно прості числа, то T – множина всіх первісних коренів ступеня mn з 1.

Доведення ми опускаємо.

Лема 4.2. Позначимо через $\sigma(k)$ суму всіх первісних корінь ступеня k з 1. Якщо m й n взаємно прості, то $\sigma(mn) = \sigma(m)\sigma(n)$.

Доведення ми опускаємо..

Лема 4.3. $\sigma(n)=\mu(n)$, де $\mu(n)$ – функція Мебіуса.

$$\mu(n)=df=\begin{cases} -1, & n=p_1p_2...p_r, r=2l+1 \\ 1, & n=p_1p_2...p_r, r=2l \\ 0, & n \text{ділиться на квадрат деякого простого числа} \end{cases}$$

Доведення ми опускаємо.

Представимо число n у вигляді добутку ступенів його різних простих дільників: $n=p_1^{k_1}...p_r^{k_r}$, покладемо $m=p_1...p_r$, $d=n/m$. Нехай $\{x_1,x_2,...,x_{\varphi(m)}\}$ – множина всіх первісних коренів ступеня m з 1; $\Phi_m(x)=\prod_{j=1}^{\varphi(m)}(x-x_j)$, де $\varphi(m)$ – функція Ейлера. Тоді має місце

Лема 4.4. Мінімальним поліномом для первісних коренів числа n є поліном $\Phi_n(x)=\Phi_m(x^d)$.

Доведення ми опускаємо.

Лема 4.5. Нехай $m=p_1\cdot...p_r$, $k=l\cdot s$, де l – число, взаємно просте з m, а $s=p_1^{w_1}\cdot...\cdot p_t^{w_t}$, $t\le r$. Якщо $\{x_1,x_2,...,x_{\varphi(m)}\}$ – множина всіх первісних коренів ступеня m з 1, то $\sum_{j=1}^{\varphi(m)}x_j^k=(p_1-1)\cdot...\cdot(p_t-1)\cdot(-1)^{r-t}$.

Доведення ми опускаємо.

За цією лемою, сума k-тих степенів залежить від того, чи є спільні дільники чисел k та m. Наведемо приклади:

Приклад 4.2. Обчислення сум k-тих степенів за лемою 4.5.

Нехай $m=30$. Оскільки $30=2\cdot3\cdot5$, $p_1=2$, $p_2=3$, $p_3=5$. $r=3$ (r – кількість простих дільників m. $\varphi(m)=8$. Число $\varphi(m)$ у даному випадку обчислюється за формулою:

$$\text{для } m=p_1p_2...p_k \quad \varphi(m)=(p_1-1)(p_2-1)...(p_k-1).$$

Степінь мінімального полінома дорівнює $\varphi(30)=8$. Отже, $\deg(\Phi_{30}(x))=8$.

1. $k=7$. $t=0$ (7 не має спільних дільників з 30). $P_7=(-1)^3=-1$.
2. $k=6$. $t=2$ (6 ділиться на 2,3). $P_6=(2-1)(3-1)(-1)^1=-2$.
3. $k=5$. $t=1$ (5 ділиться на 5). $P_5=(5-1)(-1)^2=4$.

Алгоритм побудови мінімального полінома використовує формули Вієта та формули Ньютона. Наведемо ці формули:

Формули Вієта. Позначимо через $S_k(x_1,...,x_n)$ елементарну симетричну функцію від змінних $x_1,...,x_n$, $1\le k\le n$.

$S_1=x_1+...+x_n$,

$S_2 = x_1x_2 + ...x_ix_j + ... + x_{n-1}x_n$,

…

$S_n = x_1x_2...x_n$.

Формули Вієта встановлюють зв'язок між коренями $x_1,...,x_n$ полінома $f(z) = z^n + a_1z^{n-1} + ... + a_{n-1}z + a_n$ та його коефіцієнтами $a_1,...,a_n$:

$$S_1 = -a_1, S_2 = a_2,...S_j = (-1)^j a_j,...,S_n = (-1)^n a_n$$

Формули Ньютона використовуються для обчислень значень елементарних симетричних функцій через суми k-тих степенів:

Введемо позначення:

$$P_k(x_1,...,x_n) = x_1^k + ... + x_n^k.$$

Співвідношення між функціями P_j та S_k встановлюють формули Ньютона:

$P_k - P_{k-1}S_1 + ... + (-1)^{k-1}P_1S_{k-1} + (-1)^k kS_k = 0$, якщо $1 \le k \le n$;

Основний алгоритм {Побудова мінімального полінома первісного кореня ступеня n з 1.}

Вхід: натуральне *n;*

Вихід: $m, d;\ a_0,...,a_{\varphi(m)}$ **;**

{ $f(x) = S_{\phi(m)} + S_{\phi(m)-1}x^d + ... + S_1x^{(\varphi(m)-1)d} + x^{\varphi(m)d}$ **}**

1. Для $n = p_1^{\alpha_{11}} \cdot ... \cdot p_r^{\alpha_r}$ обчислити $r,\ p_1,...,p_r;\ m = p_1 \cdot ... \cdot p_r$ **;**
2. $d = n\ div\ m$**;**
3. Обчислити $S_1 = \mu(m);\ P_1 = S_1;$
4. Для j від 2 до $\varphi(m)$ обчислити

   4.1. Обчислити $P_k = (p_{j1} - 1)(p_{j2} - 1)...(p_{jt} - 1) \cdot (-1)^{r-t}$

   4.2. Обчислити S_k з формул Ньютона.

Крок 1 можна реалізувати за модифікованим алгоритмом Ератосфена. На виході цього алгоритму – послідовність простих дільників n, їхній добуток m і число d – ступінь змінної x у формулі леми 4.4.

Крок 2 полягає в обчисленні $\mu(m)$ – парності числа r. Цей крок ініціює цикл 3.1-3.2.

Крок 3.1 обчислює суму k-тих ступенів первісних коренів. Відповідно до леми 4.5 ця сума дорівнює добутку чисел $p_j - 1$ для всіх таких j, що p_j ділить k. Ці обчислення можна реалізувати методом пробних ділень числа k на p_j. Число t дорівнює числу успіхів у циклі пробних ділень.

Крок 3.2. реалізується циклом від 1 до k, у якому обчислюються S_k.

Операція ділення у полі TCoef. Вище визначені алгоритми операцій додавання, віднімання та множення елементів поля *TCoef*. Виявлено, що нетривіальною для реалізації є операція множення двох елементів поля, які є значеннями тригонометричних поліномів виду

$$T(x) = \cos(mx) + c_1 \cos((m-1)x) + ... + c_{m-1} \cos(x) + c_m, c_j \in Q$$

при $x = 2\pi/n$. Залишилося визначити алгоритм операції ділення двох елементів *TCoef*, тобто надати алгоритм обчислення коефіцієнтів поліномна $T(x) = T_1(x)/T_2(x)$.

Оскільки поле *TCoef* є алгебраїчним розширенням поля Q, ми можемо застосовувати стандартний алгоритм, який полягає у наступному:

Знаходимо елемент *TCoef*, обернений до $T_2(x)$. Для цього застосовуємо розширений алгоритм Евкліда до поліномів $M(y), T_2(y)$, де $M(y)$ – мінімальний поліном числа $\cos(2\pi/n)$. Розширений алгоритм Евкліда разом з $\mathrm{GCD(M(y), T_2(y))}$ знаходить такі поліноми $D_1(y), D_2(y)$, що $D_1(y)M(y) + D_2(y)T_2(y) = GCD(M(y), T_2(y))$. Оскільки $M(y)$ є незвідним над Q і $\deg(T_2) < \deg(M)$, маємо $\mathrm{GCD(M(y), T_2(y))} = 1$. Отже, $D_1(y)M(y) + D_2(y)T_2(y) = 1$. Оскільки при $x = 2\pi/n$ $M(x) = 0$, $D_2(x)T_2(x) = 1$ або $D_2(x) = \dfrac{1}{T_2(x)}$.

Цей метод можна модифікувати для Т-поліномів. Для цього розглянемо тригонометричну тотожність

$$\cos(a)\cos(b) = \frac{1}{2}\cos(a+b) + \frac{1}{2}\cos(a-b).$$

Нехай $a = mx, b = (n-m)x$. Тоді

$$\cos(mx)\cos((n-m)x) = \frac{1}{2}\cos(nx) + \frac{1}{2}\cos(2m-n)x),$$

або

$$\cos(nx) = (2\cos(n-m)x))\cos(mx) - \cos((2m-n)x).$$

Ця рівність використовується у розширеному алгоритмі Евкліда для вирівнювання степенів тригонометричних поліномів, старші члени яких ми позначаємо через $\cos(nx), \cos(mx), m < n$.

Модифікований розширений алгоритм Евкліда для Т-поліномів

Вхід*: P, Q – T-поліноми від змінної x.*

$$P = c_{10}\cos(nx) + c_{11}\cos((n-1)x) + ... + c_{1n},$$

$$Q = \cos(mx) + c_{21}\cos((m-1)x) + ... + c_{2m}$$

Вихід: D, T_1, T_2 – *T-поліноми* від змінної x.

Інваріантні співвідношення $D = GCD(P,Q)$, $T_1P + T_2Q = D$.

```
ExtTrigGCD = rs(P,Q,U1,V1,U2,V2,M)
{
(P, Q) = (P,Q,1,0,0,1),    //P = 1*P + 0*Q,  Q = 0*P + 1*Q
(0,Q,U1,V1,U2,V2)=(Q,U2,V2),    //Q  = GCD(P,Q), T1 = U2, T2 = V2
Deg(P) <= Deg(Q)  →
(P,Q,U1,V1,U2,V2)=(Q-M*P, P, U2-M*U1, V2-M*V1, U1, V1),
Deg(P) > Deg(Q)  →
(P,Q,U1,V1,U2,V2)=(Q, P, U2, V2, U1, V1)
};
```

Тригонометричний одночлен M обчислюється за формулою

$$M = \frac{c_{20}}{c_{10}}\cos((n-m)x),\ n = \deg(Q), m = \deg(P). \qquad (4.13)$$

Алгоритм *ExtTrigGCD* відрізняється від розширеного алгоритму Евкліда для поліномів лише формулою (4.13) обчислення додаткового множника M та інтерпретацією множення як множення Т-поліномів.

4.2. Алгебри числових множин

Шкільний курс алгебри, тригонометрії а також основні математичні курси вищої школи використовують декілька алгебр числових множин. Нижче розглянуті специфікації алгебр одновимірних числових множин (тобто множин на числовій вісі) у термінах розширень, морфізмів та наслідування.

4.2.1. Ієрархія наслідування алгебр числових множин

Ієрархія алгебр числових множин представлена на рис. 4.1.

Сорт *Set* – абстрактна алгебра множин. Місце цієї алгебри в загальній ієрархії наслідування абстрактних алгебр визначено на рис. 4.1. Наслідування визначає сигнатури теоретико-множинних та арифметичних операцій алгебр числових множин.

Сорт *NumSet* – абстрактна алгебра одновимірних числових множин, параметризована розширеною числовою віссю – упорядкованим числовим полем *ExtCoef* та виділеними константами *PlusInf, MinusInf*, які позначають плюс та мінус нескінченність. Ці константи використовуються для того, щоб не відрізняти необмежені числові проміжки (наприклад, $(-\infty, b]$) від обмежених ($(a,b]$). Сигнатура *NumSet* включає операції

```
Inf: NumSet → Coef, Sup: NumSet; → Coef;
Min: NumSet → Coef,  Max: NumSet; → Coef.
```

та деякі інші операції над числовими множинами на числовій вісі.

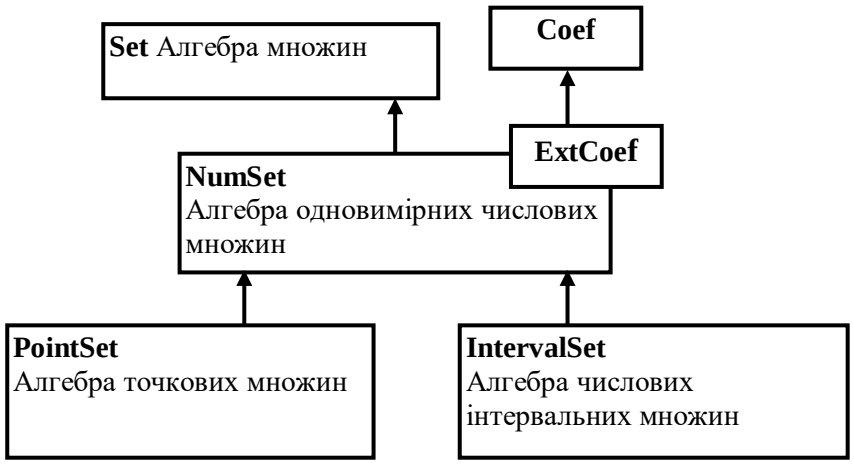


Рис. 4.1. Ієрархія наслідування одновимірних числових множин

Розширену числову вісь *ExtCoef* реалізовано наслідуванням упорядкованого поля *Coef.* В інтерпретаторах операцій та відношень порядку розширеної числової вісі *ExtCoef* для констант *PlusInf, MinusInf* визначені такі властивості:

```
a ∈ Coef → {
   PlusInf + a  = PlusInf, MinusInf + a  = MinusInf;
   PlusInf - a  = PlusInf, MinusInf - a  = MinusInf;
   a - PlusInf = MinusInf, a - MinusInf = PlusInf;
   a>0→PlusInf*a=PlusInf, a<0→ PlusInf*a=MinusInf,
   a<0→MinusInf*a=PlusInf, a>0→ MinusInf*a=MinusInf;
   a>0→PlusInf/a=PlusInf, a<0 → PlusInf/a=MinusInf,
   a>0→MinusInf/a=MinusInf, a<0→ MinusInf/a=PlusInf,
   a/PlusInf=0, a/MinusInf=0;
   MinusInf<a, a<Plusnf;
};
```

Крім теоретико-множинних операцій, в алгебрі *NumSet* визначені арифметичні операції *Add, Subsrtuct, Mult, UnionMult:*

```
Add(2):Coef×NumSet→NumSet, ac;

Subsrtuct(2):Coef×NumSet→NumSet, NumSet×Coef→NumSet;

Mult(2):Coef×NumSet→NumSet, ac;

UnionMult(2),()°():Nat×NumSet→NumSet, ac;
```

Операції *Add, Subsrtuct, Mult* використовуються для розв'язання задач типу $ax+b\in S$, де $S\in NumSet$ (числова множина). В термінах цих операції ця задача розв'язується одним правилом: $ax+b\in S=a^{-1}*(S-b)$.

Семантика операції *UnionMult* більш складна:

$$k \circ S = S \cup 2 * S \cup ... \cup k * S.$$

Ця операція є похідною від операцій *Union, Mult:*

```
UnionMult:{
   1°S = S,
   k°S = k*S∪(k-1)°S
};
```

Сорт *PointSet* – алгебра конструктивних точкових множин. Елементами конструктивно визначеної точкової множини може бути або скінченна точкова множина, або нескінченна конструктивно визначена точкова множина. Конструктивні визначення нескінченних множин мають бути реалізованими операціями

Element: Function×Nat→Coef,
NextElement: Function×PointSet→Coef.

Операція *Element* задає числову множину означенням $a_n = f(n)$, де функція f визначається означенням в специфікації конкретної алгебри.

Операція *NextElement* задає числову множину означенням $a_{n+1} = f(a_n)$, де функція f визначається означенням в специфікації конкретної алгебри. Ми допускаємо визначення числових множин будь-якими конструктивними засобами.

Сорт *IntervalSet* – алгебра конструктивних числових інтервальних множин. Елементами цієї алгебри є або скінченні об'єднання числових проміжків, або нескінченні конструктивно визначені об'єднання числових проміжків.

$$S = A_1 \cup A_2 \cup ... \cup A_k, \text{ або } S = A_1 \cup A_2 \cup ... \cup A_k \cup ...,$$

$$A_i = (a,b), \text{ або } A_i = [a,b), \text{ або } A_i = (a,b], \text{ або } A_i = [a,b]$$

Як і для точкових множин, ми визначаємо операції

Element(1):Function×Nat → Interval,

NextElement(2):Function×PointSet → Interval.

Ці функції визначають окремі інтервали у множині S.

4.2.2. Структура разширень алгебр числових множин

На рис. 4.2 представлено діаграму розширень БАС числових множин. Ця діаграма визначає конструктивні алгебри числових множин шкільного курсу алгебри, включаючи тригонометрію (періодичні числові множини).

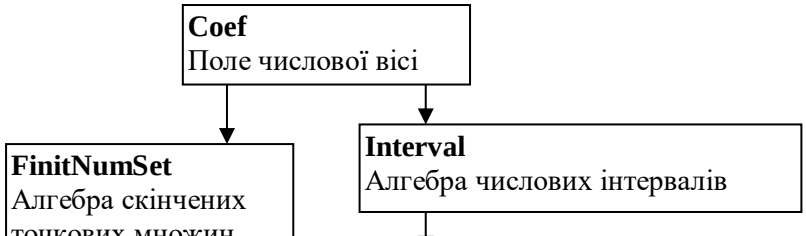


Рис. 4.2. Структура розширень одновимірних числових множин

Сорт *FinitNumSet*

```
Sort FinitNumSet::PointSet;
    Parameter Field Coef;
    Constructor
    FinitNumSet P = {
    FinitNumSet P = {Coef M, FinitNumSet Q} |
    {Coef M) | Empty;                          // Конструктор сорту
    {Empty, P} = {P};                          // Функція вкладення
    {M, Empty} = {M};                          // Функція вкладення
    LeadElement(P) = M,                        // Функції доступу
    Deg(P)= M;
    Deg(M)= M;
    M < Deg(Q);                                // Контекстна умова
    Form: M ∈ Coef, Q ∈ FinitNumSet,
     {Empty, P} = {P}, {M, Empty} = {M};
     Deg(M)> Deg(Q)
    };
```

Operations

```
    Union: {M, P}∪{M, Q}=(M, P∪Q};
    Intersect: {M, P}∩{M, Q}=(M, P∩Q};
    Substruct: {M, P}-{M, Q}=(P-Q};
    In:{
    M ∈ {M, P} = True,
    M < N → N ∈ {M, P} = N ∈ {P},
    M > N → N ∈ {M, P} = False,
    M ∈ {M} = True,
    M < N → N ∈ {M} = False,                   // N ∈ Empty
    M > N → N ∈ {M} = False,
    M ∈ Empty = False,
```

```
};
Add: a+(M, P) = (a+M, a+P);
Substruct:{
a-(M, P) = (a-M, a-P),
(M, P)-a = (M-a, P-a)
};
Mult: a*(M, P) = (a*M. a*P);
```

В специфікаціях теоретико-множинних операцій наведені лише основні правила обчислень операцій *Union, Intersect, Substract*. Неважко побачити, що алгебру *FinitNumSet* можна визначити як лінійне динамічне розширення *Coef*. Більш точно, у визначенні лінійного динамічного розширення (3.14) $B_{(0)} = \{a\} \vee Empty, a \in Coef$. Тому повні системи інтерпретуючих правил операцій *Union, Intersect, Substract* можна вивести методами гл. 3 (див. приклад 3.10), вивід операції *Add*). В специфікаціях наведено повну систему правил лише для операції *In*. Зауважимо, що її можна оптимізувати:

```
In: {
 M ∈ {M, P} = True,
 M < N → N ∈ {M, P} = N ∈ {P},
 M > N → N ∈ {M, P} = False,
 M ∈ {M} = True,
 M <> N → N ∈ {M} = False,      // Або N ∈ {M} = False
 M ∈ Empty = False
};
```

Обчислення в алгебрі *FinitNumSet* можна реалізувати на нижньому рівні, якщо використати алгебру *ALinComb* абстрактних векторних просторів. Для цього представлення елемента $P = \{s_1, s_2, ..., s_k\}$ слід перетворити до виду $P = \alpha_1 \$ s_1 + + \alpha_2 \$ s_2 + + ... + + \alpha_k \$ s_k$, $\alpha_i \in \{False, True\}$.

Сорт *Interval*. Елементами *Interval* є числові інтервали $A = (a,b)$, $A = [a,b)$, $A = (a,b]$, $A = [a,b]$. Алгебра *Interval* параметризована розширеною числовою віссю. Таким чином, до розгляду включено і нескінченні, і пусті інтервали. Операція *Unit* буде визначена частково. Її повне визначення буде надано в алгебрі *IntervalSet***.**

Основною проблемою при реалізації цієї алгебри є проблема часткових випадків. Можна підрахувати, що повна система інтерпретуючих правил для теоретико-множинних операцій, написана на основі повного перебору можливих варіантів, містила б 96 правил. Тому в специфікаціях будемо використовувати наступну діаграму статичних розширень (рис..4.3). Принцип специфікацій полягає у наступному: алгебру відкритих інтервалів *OpenInterval* ми розширюємо до алгебри *Interval*, додаючи до розгляду крайні точки.

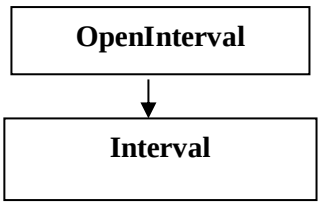


Рис 4.3 Діаграма статичних розширень

4.2.3. Структура розширень алгебри Interval

Сорт *OpenInterval* – алгебра відкритих інтервалів числової вісі. Теоретико-множинні операції алгебри – *Intersect, Union, In*. Зауважимо, що операцію *Union* визначено частково – лише для відкритих інтервалів, що перетинаються. Операція *Substract* для відкритих інтервалів не є замкненою: тип результату не співпадає з типами аргументів. Тому вона буде інтерпретованою частково в алгебрі *Interval* як похідна від операцій *Intersect, Union.* Повна інтерпретація операцій *Union, Substract* можлива лише в алгебрі *IntervalSet.* Крім теоретико-множинних операцій, визначені операції часткового порядку «менше», «більше» – відповідно *Les* та *Gre*. Вони визначені для інтервалів, що не перетинаються:

$$(a,b) < (c,d) \Leftrightarrow b \le c$$

```
Intersect:{
  Max(a,c)<Min(b,d)→(a,b)∩(c,d)=(Max(a,c),Min(b,d)),
   (a,b)∩(c,d)=Empty // Max(a,c) >= Min(b,d)
  };
Union:{Max(a,c)<Min(b,d)→(a,b)∪(c,d)=(Min(a,c),Max(b,d));
  In:x∈ a, b)=(x > a)&(x < b);
  Les:(a, b)<(c, d)=(b<=c); // Інтервали не перетинаються
  Gre:(a, b)>(c, d)=(d>=a);
  };
```

Copm Interval визначається наступною специфікацією:

Sort Interval::**IntervalSet;**

Parameter **Field ExtCoef**;

Constructor

```
Interval P = {
 // Конструктор сорту
((ExtCoef L, Bool LP),(ExtCoef R, Bool RP))|Empty;
 (L,False),(R,False))=(L,R); // Функція вкладення у OpenUnterval
(L,True),(L,True))={L};      // Функція вкладення у PointSet
LeftPoint(P)=L,              // Функції доступу
RightPoint(P)=R,
LeftClosed(P)=LP,
RightClosed(P)=RP,
 (L<R)or(L=R)&LP&RP;         // Контекстна умова
Form:{L, R ∈ Coef, LP, RP ∈ Bool, L < P
};
```

Інтерпретація монотонних операцій у алгебрі *Interval* відрізняється від їх інтерпретації у алгебрі *OpenInterval* інтерпретатором функцій *Min, Max,* яким надані імена відповідно *MinBool, MaxBool*. Розглянемо алгебру *PointBool* з носієм

```
PointBool A = (ExtField A, Bool LA)
```

та операціями

```
MinBool(2): PointBool × PointBool → PointBool,
MaxBool(2): PointBool × PointBool → PointBool,
Rev(1): PointBool → PointBool;
MaxBool:{
      a > b → MaxBool((a, u),(b, v))=(a, u),
      a < b → MaxBool((a, u),(b, v))=(b, v),
      MaxBool((a, u),(a, v))=(a, Max(u,v))
};
MinBool:{
      a < b → MinBool((a, u),(b, v))=(a, u),
      a > b → MinBool((a, u),(b, v))=(b, v),
      MinBool((a, u),(a, v))=(a, Min(u,v))
};
Rev:  Rev(a, u) = (a, Not(u));
```

Специфікації теоретико-множинних операцій алгебри *Interval* мають вигляд:

```
Les: {
((a,u1),(b,v1))<((b,u2),(d, v2))= not(v1 | u2),
((a,u1),(b, v1))<((c,u2),(a, v2))= not(v2 | u1),
((a,u1),(b, v1))<((c,u2),(d, v2))=b < c
};
Gre: ((a,u1),(b,v1)>((c,u2),(d,v2)) =
                ((c,u2),(d, v2))<((a,u1),(b, v1);
```

// Якщо інтервали не перетинаються, кожен з них лежить по одну сторону від іншого.

```
Intersect:{
 MaxBool(a,b)<MinBool(c,d)→(a,b)∩(c,d)=
                          (MaxBool(a,b),MinBool(c,d)),
 (a,b)∩(c,d)=Empty   // Max(a,b)>=Min(c,d)
};
Union:{
 MaxBool(a,c)<MinBool(b,d)→(a,b)∪(c,d)=
                          (MinBool(a,c),MaxBool(b,d)),
v1∨u2→((a,u1),(b,v1))∪((b,u2),(d,v2))=((a,u1),(d,v2))
};
Substract:{
(a,b)-(c,d)=
      (a,b)∩(MinusInf,Rev(c))∪(a,b)∩(Rev(d), PlusInf)
};
In:{
a∈((a,True),(b,v))=True,
b∈((a,u),(b,True))=True,
x∈((a,u),(b,v))=(x>a)&(x<b)
};
```

Сорт *IntervalSet.* Елементом алгебри *IntervalSet* є об'єднання скінченної упорядкованої послідовності числових інтервалів.

$$S = A_1 \cup ... \cup A_k,\ A_i = [(a_i, b_i)],\ A_i \cap A_j = \varnothing,\ b_i \le a_{i+1}$$

Подвійні скобки [(,)] означають або круглі, або прямокутні дужки. Таким чином, A_i – елементи алгебри *Interval*, що попарно не перетинаються. Неважко побачити, що алгебра *IntervalSet* є лінійним динамічним розширенням алгебри *Interval.*

```
Sort IntervalSet::NumSet;;
Parameter
Field ExtCoef;
Constructor {
IntervalSet P = Interval S++IntervalSet Q | Interval S |
Empty;
```

```
    Empty ++ P = P,
    M ++ Empty = M;
    LeadInterval(P) = S,                    // Функції доступу
    LeftClosed(P) = LeftClosed(S),
    // Контекстна умова
    (RightPoint(S) < Inf(Q)) |
    (RightPoint(S)=Inf(Q)& Not(RightClosed(S)|LeftClosed(Q));
    Form: S ∈ Interval, Q ∈ IntervalSet,
     0 ++ P = P, M ++ 0 = M,
      S <> Empty
    };
Operations
    Intersect:{
      (a∩b<>∅)→(a++A)∩(b++B)=a∩b++((a∩B)∪(b∩A))++A∩B,
    a < b → (a++A)∩(b++B)= A ∩(b++B),
    a > b → (a++A)∩(b++B)=(a++A)∩B
    }
     // Інші правила виводяться методами лінійного динамічного розширення
    Union:{
      (a∩b<>∅)→(a++A)∪(b++B)=((a∪b)∪A)∪B,
    a < b → (a++A)∪(b++B)= a++A∪(b++B),
    a > b → (a++A)∪(b++B)= b++(a++A)∪B
    }
    // Інші правила виводяться методами лінійного динамічного розширення
    Substract:{
      (a∩b<>∅)→(a++A)-(b++B)= (a-b)++((A-b)∪(a-B)∪(A-B)),
    a < b → (a++A)-(b++B)= a++(A-b)∪(A-B),
    a > b → (a++A)-(b++B)= (a++A)-B
    };
    In:  x ∈ a++A = (x ∈ a) | (x ∈ A);
```

Сорт *GenNumSet*. Елементом алгебри *GenNumSet* є об'єднання скінченної упорядкованої послідовності числових множин на числовій вісі, кожна з яких є або числовим інтервалом, або скінченною точковою множиною.

$$S = A_1 \cup ... \cup A_k,\ A_i = [(a_i, b_i)] \vee A_i = \{a_{i1}, ..., a_{im}\},\ A_i \cap A_j = \varnothing$$

Розглянемо алгебру числових множин *BaseNumSet* з носієм – об'єднанням носіїв алгебр *Interval* та *FinitNumSet:*

$$BaseNumSet = Interval \cup FinitNumSet.$$

Частинний порядок на цьому об'єднанні можна визначити таким чином:

$$A < B \Leftrightarrow \forall a \in A\ \forall b \in B\ \ a < b.$$

Зауважимо, що, з математичного погляду, алгебра *BaseNumSet* є розширенням алгебр *Interval* та *FinitNumSet,* однак вона не відповідає схемі побудови розширень, якою ми користувалися досі.

Якщо алгебру *BaseNumSet* визначено, алгебру *GenNumSet* можна визначити як лінійне динамічне розширення алгебри *BaseNumSet.* Отже, специфікації операцій алгебри *GenNumSet* не відрізняються від специфікацій операцій алгебри *IntervalSet*. Різниця полягає в інтерпретації цих операцій в алгебрі *BaseNumSet.* Таким чином, треба перевантажити теоретико-множинні операції на такі типи аргументів:

```
Intersect: Interval × FiniteNumSet → FinitNumSet, ac;

Union: Interval × FiniteNumSet → GenNumSet, ac;

Substract: FiniteNumSet × Interval → FinitNumSet;

Substract: Interval × FiniteNumSet → GenNumSet;
```

// Перетин скінченної числової множини та числового інтервалу

```
Intersect:{
a∈B → {a,A}∩B ={a, A∩B),
{a,A}∩B = A∩B
};
```

// Об'єднання скінченної числової множини та числового інтервалу

```
Union:{

a ∈ B → {a,A}∪B = A∪B,
a < LeftPoint(B) → {a,A}∪B = {a}∪A∪B,
a > RightPoint(B) → {a,A}∪B = (A∪B)∪{a}
};
```

// Скінченна числова множина мінус числовий інтервал

```
Substract{
A∩B = ∅ → A-B = A,  // Це співвідношення є абстрактним
a ∈ B → {a, A}-B = {a, A-B},
a < LeftPoint(B) → {a, A}-B = A-B
};

// Числовий інтервал B мінус скінченна числова множина A

Substract{
A∩B = ∅ → B-A = B,  // Це співвідношення є абстрактним
a < LeftPoint(B) → B-{a, A} = B-A,
```

//Число a належить числовому проміжку B = ((b,u),(c,v))

```
((b,u),(c,v))-{a,A}=
                (b,u),(a,False))++(((a,False),(c,v)-A)
};
```

4.2.4. Алгебри періодичних точкових множин

Елементами алгебри періодичних точкових множин є узагальнені арифметичні прогресії (див. п.1.1.4.).

Необхідність реалізації періодичних числових множин виникає, перш за все, при розв'язанні тригонометричних задач на рівняння та нерівності. Зауважимо, що конструкції алгебри періодичних числових множин є загальними для будь-яких базових сортів структури одновимірних числових множин. Незалежно від означення базового числового сорту *SortB*, періодичний сорт *SortA* визначається співвідношеннями

$$SortB \subset SortA;$$
$$\exists t \, \forall x((t > 0) \,\&\, (x \in SortA \to (x + t) \in SortA))$$

В структурі разширень алгебр числових множин ієрархії (рис. 4.1) представлені реріодичні сорти *PeriodicNumSet* та *PeriodicIntervalSet.*

Означення 4.4. Нехай $S \subset F$ – числова множина і t – додатний елемент дійсного поля F (тобто *ExtCoef*). Періодичною числовою множиною (S, t) з періодом t називається множина

$$(S, t) = \{s + kt : s \in S, k \in Z\}.$$

Якщо елементи множини S є елементами конкретної алгебри числових множин A, визначена алгебра називається періодичним розширенням A і позначається через (A,t). Алгебра A називається базовою алгеброю періодичного розширення (A,t).

Прикладами базових алгебр періодичних розширень *PeriodicNumSet* та *PeriodicIntervalSet* є відповідно алгебри *FinitNumSet, IntervalSet.*

Обчислення в алгебрі періодичних числових множин з фіксованим періодом t по суті не відрізняються від обчислень в базовій алгебрі. Нові риси з'являються в алгебрах, елементи яких є арифметичними прогресіями з різними періодами. Тому ми узагальнимо це означення на цей випадок.

Означення 4.5. Нехай A – (базова) алгебра числових множин і T – підмножина дійсного поля F: $T \subset F$. Періодичним розширенням A з (множинним) періодом T називається алгебра

$$(A, T) = \{s + kt : s \in S \in A, k \in Z, t \in T\}.$$

Зокрема, в шкільних тригонометричних задачах на рівняння та нерівності періоди, як правило, є величинами, кратними π. Таким чином, для цього класу задач простір періодів є одновимірним векторним простором над полем раціональних чисел з символьною константою π у якості базисного вектора. За нашим означенням, множиною T є множина додатних раціональних чисел Q. Надання константі π дійсного значення задає ізоморфізм векторного простору в поле дійсних чисел F.

Абстрактний сорт *PeriodicSet* є статичним розширенням абстрактної базової алгебри *NumSet*. Ми, однак, не будемо використовувати вкладення

NumSet в *PeriodicSet* за умови $T=\{0\}$, за винятком вкладень пустих множин. Є сенс вважати, що пуста множина є періодичною з періодом $t=0$.

Означення 4.6. Представлення елемента (S,t) алгебри (A,t) називається приведеним, якщо елемент t є найменшим додатним періодом і $\forall s \in S \in A \;\; 0 \le s < t$.

Наведене представлення елементу алгебри (A,t) є канонічною формою. Приведення елемента (S,t) до його канонічної форми по суті є операцією базової алгебри. Нижче ми представимо ці алгоритми для алгебр *FinitNumSet, IntervalSet.*

Як ми бачили ще у п.1.1.4, при реалізації теоретико-множинних операцій основним є алгоритм приведення двох елементів алгебри *PeriodicSet* до спільного періоду. Якщо ж два елементи мають спільний період, теоретико-множинні операції виконуються над базовими множинами:

$$(A,t)\cap(B,t)=(A\cap B,t),(A,t)\cup(B,t)=(A\cup B,t),(A,t)-(B,t)=(A-B,t).$$

Зауважимо, що після виконання теоретико-множинних операцій, результат, взагалі кажучи, треба приводити.

Означення 4.7. Найменшим спільним періодом періодів $t_1, t_2 \in T$ називається такий найменший елемент $t \in T$, що для деяких натуральних m, n виконується рівність $t = nt_1 = mt_2$.

Нехай $t_1 = \alpha t_b, t_2 = \beta t_b, \alpha, \beta \in Q$, де t_b – базисний елемент T. Обчислимо дріб $m/n = \alpha/\beta$. Чисельник m та знаменник n цього дробу є шуканими числами. Назвемо їх додатковими множниками періодів t_2, t_1. Процедуру обчислення найменшого спільного періоду t_1, t_2 позначимо через $LCP(t_1,t_2)$. Ця процедура повертає $m, n, t = nt_1 = mt_2$. Таким чином, $LCP(t_1,t_2)=(t,m,n).$

Алгебра PeriodNumSet. Розглянемо процедуру приведення елемента (S,t), $S \in FinitNumSet$. Вона заснована на операції $Frac(s,t)$ обчислення дробової частини та операції $Entier(s,t)$ цілочислового ділення двох додатних чисел s і t. Ці операції задаються співвідношеннями

$$s = Entier(s,t)\cdot t + Frac(s,t),\ Entier(s,t) \in N,\ 0 \le Frac(s,t) < t \qquad (4.14)$$

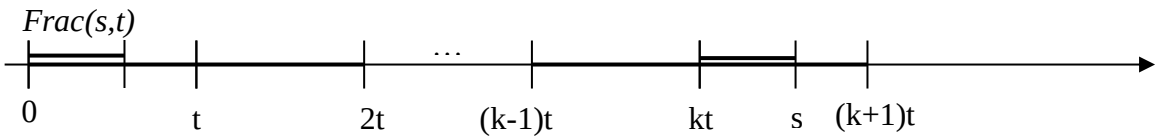


Рис 4.4. Ціла та дробова частини від цілочислового ділення двох додатних чисел. $k = Entier(s,t),\ s - kt = Frac(s,t)$

Рис 4.4 є графічною ілюстрацією обчислення операцій $Entier(s,t)$, $Frac(s,t)$ на числовій осі. Співвідношення (4.14) можна узагальнити для від'ємних чисел s.

$$s = Entier(s,t)\cdot t + Frac(s,t),\ Entier(s,t) \in Z,\ 0 \le Frac(s,t) < t$$

Неважко побачити, що $(s,t) = (Frac(s,t),t)$. Отже, для елемента $S \in FinitNumSet$ приведеним представленням є елемент $(S',t) = \{(s',t) : s' = Frac(s,t)\}$, який ми позначатимемо через $(Frac(S,t),t)$. Зауважимо, що при приведенні множини S деякі з приведених елементів S можуть співпасти: якщо $s_1, s_2 \in S$ і $Frac(s_1 - s_2, t) = 0$, тобто $s_1 - s_2 = kt, k \in Z$, $Frac(s_1,t) = Frac(s_2,t)$.

В результаті приведення елемента (S,t) до виду $(Frac(S,t),t)$ канонічну форму (S,t), взагалі кажучи, ще не побудовано. Справа у тому, що множина $S' = Frac(S,t)$, може бути, у свою чергу, періодичною. Позначимо через $S + t$ множину $\{q : q = s + t, s \in S\}$. Припустимо, що

$$S' = S_1 \cup S_2 \cup ... \cup S_k,\ k > 1,\ S_i \cap S_j = \varnothing,\ S_i + t = S_{i+1},\ i = 1,...k-1.$$

Тоді множину S' природно назвати періодичною. Неважко побачити, що у цьому випадку $(S',t) = (S_1,\ t/k)$, і ми побудували представлення (S,t) з періодом меншим, ніж t. На практиці, однак, для порівняння елементів $(S_1,t_1) = (S_2,t_2)$ простіше використовувати алгоритм приведення цих елементів до найменшого спільного періоду. Припустимо, що елементи $(S_1,t_1), (S_2,t_2)$ приведені:

$$S_1 = Frac(S_1,t_1), S_2 = Frac(S_2,t_2).$$

Через $k \circ S$ позначимо множину $S \cup 2\cdot S \cup ... \cup k\cdot S$. Тоді, якщо

$$(t,m,n) = LCP(t_1,t_2),\ (S_1,t_1) = (n \circ S_1, t),\ (S_2,t_2) = (m \circ S_2, t).$$

Отже, $(S_1,t_1) \equiv (S_2,t_2) \Leftrightarrow (n \circ S_1, t) = (m \circ S_2, t)$, де знак "≡" означає рівність елементів алгебри (A,t) **(**семантичну рівність), а знак **"="** – синтаксичну рівність.

Алгебра PeriodIntervalSet. Процедура приведення елемента (S,t), $S \in IntervalSet$ також використовує операції $Entier(s,t), Frac(s,t)$. Справді, розглянемо числовий інтервал – елемент алгебри *Interval* як нескінченну множину точок числової осі, до кожної з яких треба привести. Нехай $S = [(a,b)]$. Тоді періодична множина визначається формулою

$$(S,t) = \bigcup_{k=0}^{\infty} [(a+kt, b+kt)].\ \text{Якщо}\ |b-a| > t,\ (S,t) = (-\infty,\infty) = R.$$

При $|b-a|=t, S=[a,b)\vee S=(a,b]\vee S=[a,b]\ (S,t)=(-\infty,\infty)=R$. Лише при $|b-a|=t, S=(a,b)$ або $|b-a|<t$ процедура дає нетривіальний результат. В цих випадках має місце співвідношення $(S,t)=([(a+kt,b+kt)],t),\ k\in Z)$. Отже, при $frac(a,t)<frac(b,t)$ $(S,t)=([(Frac(a,t),Frac(b,t)],t)$ (див. рис. 4.5).

Як і в алгебрі *PeriodNumSet,* приведення базової множини S ще не визначає канонічну форму, оскільки приведена множина може бути періодичною.

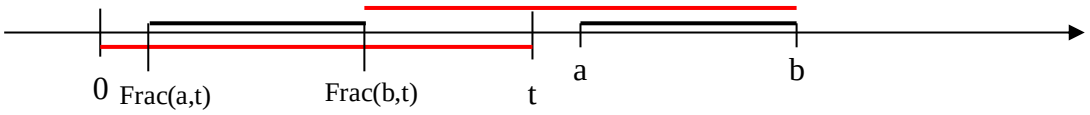


Рис 4.5. Приведення числового інтервалу по періоду *t*. Випадок $frac(a,t)<frac(b,t)$.

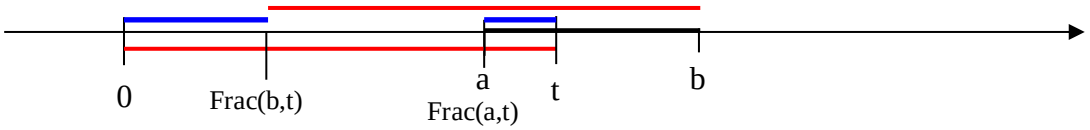


Рис 4.6. Приведення числового інтервалу по періоду *t*. Випадок $frac(a,t)>frac(b,t)$.

При $frac(a,t)>frac(b,t)$ приведена множина є об'єднанням двох числових інтервалів: $(S,t)=[0,Frac(b,t))]\cup[(Frac(a,t),t),t)$ (див. рис. 4.6).

```
Sort PeriodicSet::NumSet;
```

Parameter

NumSet BaseSet, Field PeriodSet;

```
Constructor {
 PeriodicSet S = (BaseSet A, PeriodSet t);
 (A, 0) = A, (Empty, t) = Empty;
 BSet(A, t) = A,                    // Функції доступу
 Period(A, t) = t;
 T > 0;                             // Контекстна умова
Form: A ∈ BaseSet, t ∈ PeriodSet,
 t > 0
};
```

Signature

LCP(2;3): PeriodSet×PeriodSet → Int×Int×PeriodSet;

Entier(2): PeriodSet×PeriodSet → Int;

Frac(2): PeriodSet×PeriodSet → PeriodSet;

Operations

```
Intersect:{
  (A, tA)∩(B, tB) =((A, B), LCP(tA, tB)),
(A,B), (m,n,t))= (n°A∩m°B, t)
};
```

// Інші правила виводяться методами статичного розширення

```
Union:{
  (A, tA)∪(B, tB) =((A, B), LCP(tA, tB)),
(A,B), (m,n,t))= (n°A∪m°B, t)
};
```

// Інші правила виводяться методами статичного розширення

```
Substract:{
(A, tA)- (B, tB) =((A, B), LCP(tA, tB)),
(A,B), (m,n,t))= (n°A-m°B, t)
};
In: a in (A, tA)= Frac(a, tA) in Frac(A, tA);
```

4.3. Алгоритм розв'язання тригонометричного рівняння

Розглянемо математичну модель тригонометричного рівняння (рис. 4.7).

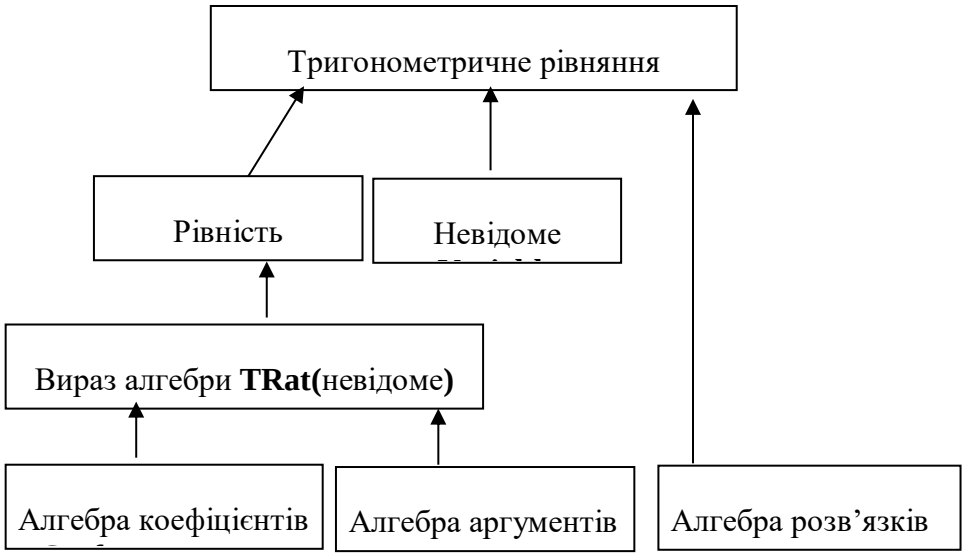


Рис 4.7. Алгебраїчний тип тригонометричного рівняння.

Алгебраїчним типом алгебраїчної задачі будемо називати її алгебраїчну модель, що визначає її як структурований алгебраїчний об'єкт з визначеннями алгебраїчних типів його елементів.

Алгоритм *Solve* розв'язання алгебраїчної задачі зводить алгебраїчний вираз – модель цієї задачі, тобто елемент алгебри *Cond*, в елемент алгебри розв'язків *Sol:*

$$Solve : Cond \to Sol.$$

Алгебраїчні типи алгебраїчних задач визначаються алгеброю умов *Cond*, алгеброю розв'язків *Sol* та розмірністю. Точне визначення алгебри розв'язків зумовлює той чи інший алгоритм *Solve*.

Приклад 4.1. Розв'язати квадратне рівняння $ax^2 + bx + c = 0$.

1. a,b,c – раціональні числа ($TCoef = Rat$). Розв'язок шукати в алгебрі

а) $Sol = Rat$ (знайти раціональні розв'язки);

б) $Sol = Rad$ (знайти розв'язки у радикалах);

в) $Sol = RatComplex$; (знайти розв'язки у полі комплексних чисел з раціональними коефіцієнтами);

г) $Sol = \mathrm{Re}al(n)$ (знайти наближені дійсні розв'язки з точністю n знаків після коми).

2. a,b,c – елементи поля $TCoef = Rad(p)$. Розв'язок шукати в алгебрі

б) $Sol = Rad$ (знайти розв'язки у радикалах);

в) $Sol = RadComplex$; (знайти розв'язки у полі комплексних чисел з радикальними коефіцієнтами);

г) $Sol = \mathrm{Re}al(n)$ (знайти наближені дійсні розв'язки з точністю n знаків після коми).

Для шкільних тригономеричних рівнянь $Cond = TrigRat(x)$, $Dim = 1$, $Sol = PeriodNumSet(Rad)$. Вказані алгебри, у свою чергу, визначаються типами алгебр коефіцієнтів $TCoef$ та $TArg$. Алгоритми обчислень у цих алгебрах розглянуті у п.4.1, 4.2.

4.3.1. Загальний алгоритм розв'язання алгебраїчної задачі

Загальний підхід до розв'язання алгебраїчних задач, який ми розглядаємо, є традиційним. Він полягає у тому, що в програмній системі для кожного алгебраїчного типу задач визначаються та реалізуються, по–перше, найпростіші задачі, по–друге, методи зведення алгебраїчних задач до найпростіших. Перелік найпростіших задач визначає потенційні функціональні можливості системи, а методи зведення – методичні можливості системи.

Нехай $P_1,...,P_k$ – логічні формули найпростіших алгебріїчних задач.

Алгоритм розв'язання задачі

Вхід: Алгебраїчна задача P.

1. Звести P до виду $F(P_1,...,P_k)$, де F – позитивна формула прикладної логіки предикатів в сигнатурі логічних зв'язок $(\&,\vee)$ і предикатів ($P \sim \mathrm{F}(P_1,...,P_k)$).

2. Тоді $Solve(P) = F(Solve(P_1),...,Solve(P_k))$, де $\&,\vee$ інтерпретуються відповідно як перетин та об'єднання розв'язків $Solve(P_j) \in Solve$.

Математична модель задачі шкільного курсу алгебри. *Модель* є безкванторною формулою прикладної логіки предикатів $F(x_1,...,x_n)$. Атомарними предикатами формули $F(x_1,...,x_n)$, є предикати рівності й заперечення рівності $(\neq)$, строгого й нестрогого порядку, а також інші атомарні предикати, визначення яких здійснюється у рамках відповідного навчального модуля. Логічні зв'язки – кон'юнкція та диз'юнкція.

Розглянемо приклад застосування цього алгоритму.

Приклад 4.2. Знайти додатні розв'язки алгебраїчного рівняння

$$\frac{2x+1}{x-2} = \frac{x-4}{x}.$$

Висхідна математична модель: $F(x) = (\frac{2x+1}{x-2} = \frac{x-4}{x}) \& (x > 0).$

Послідовність перетворень моделі:

1. $(\frac{2x+1}{x-2} - \frac{x-4}{x} = 0) \& (x > 0).$
2. $(\frac{(2x+1)x-(x-2)(x-4)}{(x-2)x} = 0) \& (x > 0).$
3. $(\frac{x^2+7x-8}{(x-2)x} = 0) \& (x > 0).$

Найпростіші задачі

4. $(x^2+7x-8=0) \& (x-2 \neq 0) \& (x \neq 0) \& (x > 0).$

Розв'язки найпростіших задач

5. $(x^2+7x-8=0) \& (x \neq 2) \& (x \neq 0) \& (x > 0).$
6. $((x=1) \vee (x=-8)) \& (x \neq 2) \& (x \neq 0) \& (x > 0).$

Спрощення:

7. $(x=1) \& (x \neq 2) \& (x \neq 0) \& (x > 0) \vee (x=-8) \& (x \neq 2) \& (x \neq 0) \& (x > 0)$

Обчислення:

8. $(x=1) \& (1 \neq 2) \& (1 \neq 0) \& (1 > 0) \vee (x=-8) \& (-8 \neq 2) \& (-8 \neq 0) \& (-8 > 0)$
9. $(x=1) \& True \& True \& True \vee (x=-8) \& True \& True \& False$
10. $x = 1$
11. $x \in \{1\}$ Представлення в алгебрі Sol.

4.3.2. Алгоритм розв'язання тригонометричних рівнянь

Типовими шкільними тригонометричними задачами є задачі на спрощення тригонометричних виразів, доведення тригонометричних тотожностей, розв'язання тригонометричних рівнянь та нерівностей [64].

Алгоритми розв'язання тригонометричних задач на спрощення та доведення тотожностей наведені у п. 4.1. Ми розглянемо алгоритм розв'язання тригонометричних рівнянь, побудований відповідно до загального підходу.

1. Найпростіші тригонометричні рівняння

1. $Sin(x) = p$, $Sin(x) = 0$.
2. $Cos(x) = p$, $Cos(x) = 0$.
3. $Tg(x) = p$, $Tg(x) = 0$.
4. $Ctg(x) = p$, $Ctg(x) = 0$.
5. $Sin(kx) = Sin(lx)$.
6. $Cos(kx) = Cos(lx)$.
7. $Tg(kx) = Tg(lx)$.
8. $Ctg(kx) = Ctg(lx)$.
9. $aCos(x) + bSin(x) = p$.

Зауважимо, що найпростіше рівняння 9 по суті є загальним випадком. Однак, у шкільному курсі тригонометрії розглядаються саме такі найпростіші рівняння.

2. Методи розпізнання типів тригонометричних рівнянь

Якщо потрібно не лише розв'язати тригонометричне рівняння, а і побудувати хід його розв'язання, методи зведення, по–перше, мають бути адекватними до тих, що використовуються у шкільному курсі тригонометрії, по–друге, мають спиратися на відомі методи тригонометричних обчислень шкільного курсу алгебри. Цю задачу ми не розглядаємо. Методам генерації ходу розв'язання задач присвячено книгу [65]. Для задачі обчислення розв'язків тригонометричного рівняння зведення полягає у.

1) побудові канонічної форми умови (п.4.1);
2) розпізнанні типів найпростіших рівнянь за канонічною формою;
3) застосуванні системи правил побудови розв'язків найпростіших рівнянь за їх типами ($Cond \to Sol$);
4) обчисленні загальної відповіді в алгебрі Sol.

Розпізнання типів найпростіших рівнянь здійснюється системою правил переписування, яка для типів рівнянь 1-9 має такий вигляд:

```
rsTrigSolve := rs(x, a, b, p)(
/* 1. Sin(x) = 0, Sin(x) = p. */
 Fi(0,1,x) = x in (0,Pi),
 Abs(p)<=1 →
   Fi(0,1,x)- p = x in ((arcsin(p), Pi-arcsin(p)),2$Pi),
/* 2. Cos(x) = 0, Cos(x) = p. */
```

```
Fi(1,0,x) = x in ((1//2$Pi, -1//2$Pi),2$Pi),
Abs(p)<=1 →
  Fi(1,0,x)- p = x in ((arccos(p), -arccos(p)),2*Pi),
```

… /* рівняння типів 1.3 - 1.8 */

```
/*  9. a·Cos(x)+b·Sin(x)=p  */
 Abs(p/Sqrt(a^2+b^2))<=1 →
   Fi(a,b,x)- p = x in (arcFi(a,b,p), 2*Pi)
);
```

Функції `arccos(p), arcsin(p), arcFi(a,b,p)` повертають розв'язки рівнянь $Cos(x)=p$, $Sin(x)=p$, $a\cdot Cos(x)+b\cdot Sin(x)=p$.

Дещо складнішою є задача розпізнання типів тригонометричних рівнянь, що зводяться до квадратних.

3. *Метод інваріантів* розпізнання рівняння, що зводиться до квадратного. Метод, який ми опишемо, має розпізнавати рівняння типу $au^2+bu+c=0$, де $u=Cos(x)$ або $u=Sin(x)$ або $u=Tg(x)$. Канонічна форма тригонометричного моному і формула множення мономів мають вигляд

$$Fi(a,b,x)=aSin(x)+bCos(x),$$

$$Fi(a,b,x)Fi(c,d,y)=Fi(\frac{ac-bd}{2},\frac{bc+ad}{2},x+y)+Fi(\frac{ac+bd}{2},\frac{bc-ad}{2},x-y)$$

Розглянемо рівняння виду $Au^2+Bu+C=0$, де $u=Fi(a,b,x)$. Застосувавши ці формули, отримаємо $(Fi(a,b,x))^2=Fi(\frac{a^2-b^2}{2},ab,2\cdot x)$,

$$Au^2+Bu+C=Fi(A\frac{a^2-b^2}{2},Aab,2\cdot x)+Fi(Ba,Bb,x)+C. \qquad (4.15)$$

Таким чином, якщо тригонометричне рівняння зводиться до квадратного, канонічна форма його лівої частини має вигляд

$$Fi(a_1,b_1,\cdot y)+Fi(a_2,b_2,x)+C \qquad (4.16)$$

де

$$\begin{cases} a_1=A(a^2-b^2)/2, \\ b_1=Aab, \\ a_2=Ba, \\ b_2=Bb, \\ y=2x \end{cases} \qquad (4.17)$$

Змінні $A,\ B,\ a,\ b$ є невідомими. Якщо виключити їх із системи (4.17), ми отримаємо співвідношення між змінними $a_1,\ b_1,\ a_2,\ b_2$, які мають виконуватися, якщо вихідне рівняння зводиться до виду (4.15) Неважко обчислити, що з (4.15) випливає

$$a_1 a_2 b_2 = b_1(a_2^2 - b_2^2),\ y = 2x. \tag{4.18}$$

Співвідношення (4.18) будемо називати умовним інваріантом задачі. Формула (4.16) визначає так званий структурний інваріант задачі. Таким чином, алгоритм розпізнавання застосовності методу зведення тригонометричного рівняння до квадратного полягає у перевірці структурного та умовного інваріантів. Відповідний фрагмент алгебраїчної програми є умовним правилом

$$(a1 * a2 * b2 = b1 * (a2\text{^}2 - b2\text{^}2)) \,\&\, (y = 2 * x) \rightarrow$$
$$Fi(a1, b1, y) + Fi(a2, b2, x) + C = TrigEquation12(a1, b1, a2, b2, x, C)$$

Така форма програми розпізнавання є стандартною для досить широкого класу задач на розпізнавання приналежності виразу деякому класу виразів:

$$< \textit{Умовний інваріант} > \rightarrow < \textit{Структурний інваріант} > = < \textit{Метод} > \tag{4.19}$$

На метод розпізнавання типів алгебраїчних виразів, який ми щойно описали, ми будемо посилатися як на *метод інваріантів*.

Зауважимо, що рівняння типу $au^2 + bu + c = 0$, де $u = Tg(x)$ – алгоритм побудови канонічної форми, зводить до виду

$$(aCos^2(x) + bCos(x)Sin(x) + cSin^2(x) = 0) \,\&\, (Cos(x) \neq 0),$$

а потім до виду $(Fi(a_1, b_1, 2x) = 0) \,\&\, (Fi(0,1, x) \neq 0)$ (до найпростішого рівняння).

Спеціальна тригонометрична факторизація. Переліки типів найпростіших тригонометричних рівнянь та методів зведення дозволяють реалізувати програмний модуль *Solver*, який розв'язує 75–80% тригонометричних рівнянь з відомого шкільного задачника [64]. Метод спеціальної тригонометричної факторизації дозволив підвищити цей показник до 90–95%. Сутність методу полягає у наступному: за канонічною формою тригонометричного рівняння $F(x) = 0$ метод намагається виділити у виразі $F(x)$ фактор $G(x)$ спеціального виду. Ми розглянемо такі спеціальні фактори:

$$G(x) = Cos(kx),\ G(x) = Sin(kx),\ G(x) = Cos(kx) - p,\ G(x) = Sin(kx) - p,$$

де

$$p \in \{\pm\frac{1}{2}, \pm\frac{\sqrt{2}}{2}, \pm\frac{\sqrt{3}}{2}, \pm 1\}. \tag{4.20}$$

Оскільки шкільні тригонометричні рівняння «полюбляють» корені вказаного виду, такий прийом дозволяє значно розширити клас розв'язуваних рівнянь. Опишемо цей алгоритм.

1. Привести рівняння до канонічної форми.
2. Заміною змінної-аргументу звести усі тригонометричні аргументи канонічної форми по вигляду $m \cdot y$, де $m-$ натуральне число.
3. До рівняння $F(x) = 0$ застосувати алгоритм, викладений нижче.

Алгоритм спеціальної тригонометричної факторизації

Алгоритм наведено для випадку, коли спеціальним тригонометричним фактором $F(x)$ є тригонометричний одночлен $\cos(kx)$. Інші випадки з (4.20) розглядаються аналогічно. Розглянемо тригонометричний вираз виду:

$$F(x) = \sum_{j=0}^{n} (a_j \cos(jx) + b_j \sin(jx)).$$

З'ясуємо наступне питання: яким умовам мають задовольняти коефіцієнти a_j та b_j, щоб вираз $F(x)$ можна було б представити у вигляді $F(x) = \cos(kx)G(x)$, де k – деяке натуральне число, а $G(x)$ – також Т-поліном $G(x) = \sum_{j=0}^{m} (u_j \cos(jx) + v_j \sin(jx))$. Можна показати, що мають місце співвідношення

1. при $m - k \geq 2k$:

$$\begin{cases} a_0 = \frac{u_k}{2}, b_0 = 0; \\ a_t = \frac{u_{k-t} + u_{k+t}}{2}, b_t = -\frac{v_{k-t} + v_{k+t}}{2}, 1 \leq t \leq k-1; \\ a_k = u_0 + \frac{u_{2k}}{2}, b_k = 0; \\ a_r = \frac{u_{r-k} + u_{r+k}}{2}, b_r = \frac{v_{r-k} - v_{r+k}}{2}, k+1 \leq r \leq 2 \cdot k; \\ a_r = \frac{u_{r+k}}{2}, b_r = \frac{v_{r+k}}{2}, 2 \cdot k < r \leq m-k; \\ a_r = \frac{u_{r-k}}{2}, b_r = \frac{v_{r-k}}{2}, m-k < r \leq m+k. \end{cases}$$

2. при $2 \cdot k > m - k$:

$$\begin{cases} a_0 = \dfrac{u_k}{2}, b_0 = 0; \\ a_t = \dfrac{u_{k-t} + u_{k+t}}{2}, b_t = -\dfrac{v_{k-t} + v_{k+t}}{2}, 1 \le t \le k-1; \\ a_k = u_0 + \dfrac{u_{2k}}{2}, b_k = 0; \\ a_r = \dfrac{u_{r-k} + u_{r+k}}{2}, b_r = \dfrac{v_{r-k} - v_{r+k}}{2}, k+1 \le r \le m-k; \\ a_r = \dfrac{u_{r-k}}{2}, b_r = \dfrac{v_{r-k}}{2}, m-k < r \le m+k. \end{cases}$$

3. при $k \ge m$:

$$\begin{cases} a_r = \dfrac{u_{r-k}}{2}, b_r = \dfrac{v_{r-k}}{2}, k-m < r \le k; \\ a_r = \dfrac{u_{r+k}}{2}, b_r = \dfrac{v_{r+k}}{2}, k \le r \le m+k. \end{cases}$$

Ці співвідношення є системою лінійних рівнянь відносно змінних u_k, v_k. Неважко скласти алгоритм з часовою складністю $O(m^2)$ розв'язання цієї системи. Якщо система є сумісною, відповідний фактор знайдено. Аналогічні формули треба вивести і для інших спеціальних факторів із (4.20).

Обчислення треба виконувати у циклі по m від 1 до n, знаходячи таким чином усі спеціальні тригонометричні фактори цілого тригонометричного виразу.

4.4. Висновки

1. Фрагмент ієрархії багатосортних алгебр, призначений для реалізації розв'язань тригонометричних задач, описано у термінах наслідування, розширень та морфізмів, тобто у відповідності до парадигми алгебраїчних обчислень.
2. Алгебраїчні обчислення використовують спеціальні канонічні форми кільця тригонометричних виразів, поля числових коефіцієнтів та алгебри періодичних числових множин.
3. Обчислення у полі числових коефіцієнтів *TCoef* спираються на спеціально розроблені алгоритми комп'ютерної алгебри – обчислення у полях ділення круга.
4. Ієрархія алгебр числових множин реалізує обчислення, необхідні для розв'язання алгебраїчних та тригонометричних рівнянь та нерівностей.
5. Наведені методи та алгоритми дозволяють розв'язувати практично усі стандартні тригонометричні задачі шкільного курсу тригонометрії.

6. Аналогічно будуються алгоритми розв’язання алгебраїчних задач інших алгебраїчних типів. Однак, кожен з таких типів потребує розробки спеціальних алгоритмів, основаних на побудові та аналізі канонічних форм.
7. Усі алгоритми розділу можуть бути ефективно реалізованими засобами системи алгебраїчного програмування АПС.

Глава 5. Основні алгоритми комп'ютерної алгебри

5.1. Множення поліномів

Задача множення поліномів та багаторозрядних чисел є однією з важливих практичних задач комп'ютерної алгебри. Множення поліномів $f(x), g(x) \in Q[x]$, $\deg(f) \le n, \deg(g) \le n$, «за означенням» потребує $O(n^2)$ арифметичних операцій з цілими числами. Справді, якщо

$$f(x) = \sum_{0 \le k \le m} a_k x^{m-k}, \quad g(x) = \sum_{0 \le l \le n} b_l x^{n-l},$$

$$h(x) = f(x)g(x) = \sum_{0 \le j \le m+n} c_j x^{m+n-j}, \text{ то } c_j = \sum_{\substack{k+l=j \\ 0 \le k \le m \\ 0 \le l \le n}} a_k b_l,$$

тобто кожен коефіцієнт $f(x)$ треба помножити на кожен коефіцфєнт $g(x)$.

5.1.1. Метод Карацуби

Алгоритм множення поліномів арифметичної складності меншої, ніж $O(n^2)$, запропонував А. Карацуба [66]. Будемо вважати, що

$$\deg(f) = \deg(g) = n,\ n = 2^k - 1. \tag{5.1}$$

Нехай $f = Ay + B,\ g = Cy + D.$ Тоді $fg = ACy^2 + (AD + BC)y + BD.$ Покладемо

$$M_1 = AC,\ M_2 = (A + B)(C + D),\ M_3 = BD.$$

Тоді $AD + BC = M_2 - M_1 - M_3$,

$$f \cdot g = M_1 y^2 + (M_2 - M_1 - M_3)y + M_3 \tag{5.2}$$

Покладемо $d = (n+1)/2 = 2^{k-1}$. Поліноми f, g представимо у вигляді

$f = (a_0 x^{d-1} + \ldots + a_{d-1})x^d + (a_d x^{d-1} + \ldots + a_{2d-1})$,

$g = (b_0 x^{d-1} + \ldots + b_{d-1})x^d + (b_d x^{d-1} + \ldots + b_{2d-1})$.

$A = a_0 x^{d-1} + \ldots + a_{d-1},\ B = a_d x^{d-1} + \ldots + a_{2d-1}$,

$C = b_0 x^{d-1} + \ldots + b_{d-1}, D = b_d x^{d-1} + \ldots + b_{2d-1}$,

$y = x^d$.

Зауважимо, що поліноми $M_1 y^2, (M_2 - M_1 - M_3)y, M_3$ не мають подібних мономів. Тому операція додавання головних доданків у (5.2) виконується за час $O(1)$, а операції віднімання у другому доданку – за час $O(n)$.

Обчислення значень M_1, M_2, M_3 є рекурсивними, причому степені поліномів при рекурсивних викликах алгоритму множення зменшуються

вдвічі. Якщо позначити через $T(n)$ складність алгоритму за часом, час обчислення T_{M_j} значення M_j дорівнює $T(n/2)$. Тому обчислення за формулою (5.2) задовольняє співвідношенню

$$T(n) = 3T(n/2) + Cn.$$

Звідси

$$T(n) = O(n^{\log_2 3}) \approx O(n^{1,59}). \qquad (5.3)$$

Зауважимо, що припущення (5.1) завжди можна виконати, якщо «дописати» до поліномів f, g бракуючу кількість старших мономів з нульовими коефіцієнтами. При цьому у (5.3) збільшиться лише мультиплікативна константа.

Цей метод застосовний також для множення багаторозрядних цілих чисел, представлених в позиційній системі числення. Якщо $f = a_0 p^n + a_1 p^{n-1} + ... + a_n$, $g = b_0 p^n + b_1 p^{n-1} + ... + b_n$, єдиною особливістю задачі множення чисел є необхідність враховувати одиниці переносів з молодших розрядів до старших. Оскільки кількість таких переносів не перебільшує кількості додавань цифр, загальна складність алгоритму множення багаторозрядних чисел також оцінюється формулою (5.3).

Алгебраїчний підхід. Стандартним представленням даних в алгоритмі Карацуби є представлення поліномів векторами коефіцієнтів. Достоїнством цього представлення є наявність та використання прямого доступу до даних. Недолік – можливо, велика кількість нулів у векторах коефіцієнтів. У цьому випадку алгоритм здійснює багаторазове множення на нуль й додавання з нульовими векторами. Представлення ж у вигляді списку (коефіцієнт – степінь) для метода Карацуби є неприйнятним. Логічно було б представляти поліноми у виді пари (старша половина, молодша половина), оскільки саме у цьому виді алгоритм використовує дані.

У термінах гл.3 це представлення є бінарним динамічним розширенням основного поля $Coef = Q$. Розглянемо його детальніше. Нехай

$$A = a_0 x^{d-1} + ... + a_{d-1},\ B = a_d x^{d-1} + ... + a_{2d-1},\ y = x^d,\ f = Ay + B.$$

Алгебру (сорт *BinPol*) визначимо наступним чином:

```
Sort BinPol::Polynom[Field Coef];
Constructor{
   BinPol f = BP(BinPol A,B; Nat k); // семантика f = Ax^(2^k) + B
   BP(0,B,k) = B;                    //функція вкладення
   MajPol(f) = A, MinPol = B, IndPol = k; //функції доступу
   IndPol(f)<max(IndPol(A),IndPol(B)), //контекстні умови
```

```
A in Q → Ind(A)=0, B in Q → Ind(B)=0
};
```

Operations

```
Mult:{
```

/* Частині випадки основного правила використовують 2 умноження */

```
  BP(A,0,k)*BP(C,D,k) = BP(A*C,BP(A*D,0,k),k+1),
  BP(A,B,k)*BP(C,0,k) = BP(A*C,BP(B*C,0,k),k+1),
```

/* Основне правило – функція у правій частні використовує 3 умноження */

```
  BP(A,B,k)*BP(C,D,k) = fnKarMult(A,B,C,D,k),
```

/* Похідні правила використовують 2 множення */

```
  B*BP(C,D,k) = BP(B*C,B*D,k),
  BP(A,B,k)*D = BP(A*D,B*D,k)
};
fnKarMult := proc(A,B,C,D, k)loc(M1,M2,M3)(
  M1 := A*C;
  M2 := B*D;
  M3 := (A+B)*(C+D);
  return(BP(M1, BP(M3-M1-M2,M2,k),k+1)
);
```

Лінійні операції у цьому кільці виконуються покоординатно:

```
  BP(A,B,k) + BP(C,D,k) = BP(A + C,B + D,k),
  a*BP(A,B,k) = BP(a*A,a*B,k).
```

Залишилося визначити алгоритм перетворення поліному з виду $f = a_0 x^n + a_1 x^{n-1} + ... + a_n$ в елемент кільця *BinPol*. Нетривіальна частина цього алгоритму – перетворення степеня x^m в елемент *BinPol*. Представимо x^m у виді $x^r \cdot x^{2^k}$, де $r < 2^k$. Тоді $x^m = BP(x^r, 0, k)$, і алгоритм рекурсивно застосовується до x^r. Іншими словами, m треба представити у двійковому коді: $m = 2^{k_1} + 2^{k_2} + ... + 2^{k_l}$, $k_1 > k_2 > ... > k_l$. Реалізацію цього алгоритму ми пропонуємо читачу у якості вправи.

5.1.2. Метод Шенхаге – Штрассена

Алгоритм множення поліномів і багаторозрядних чисел арифметичної складності $O(n \log_2 n)$ розроблено у 1971 р. [67, 68]. Він заснований на новій для нас ідеї представлення полінома набором значень на спеціально обраному наборі точок.

Нехай $f(x) = a_o x^n + a_1 x^{n-1} + ... + a_n$. Виберемо на числовій осі набір точок (x_0, x_1, x_n) і обчислимо значення $y_j = f(x_j), j \in 0..n$. Інтерполяційна теорема

Лагранжа стверджує, що для будь-якого набору пар $(x_0, y_0),(x_1, y_1),...,(x_n, y_n)$, $j \neq k$, $x_j \neq x_k$ існує єдиний поліном $f(x)$ степені, меншою або рівною n такий, що $y_j = f(x_j), j = 0,1,...,n$. Інтерполяційна формула Лагранжа [7] задає цей поліном в явному вигляді. Це й означає, що поліноми можна представляти наборами своїх значень.

Нехай дано поліноми $f(x), \deg f = k, g(x), \deg g = m$. Тоді поліном $h(x) = f(x)g(x)$ має ступінь $m+k$. Отже, якщо задати $f(x), g(x)$ наборами значень в $k+m+1$ точці, можна обчислити набір значень їх добутку $h(x)$, за яким знайти його коефіцієнти (відновити $h(x)$) . Ось ці обчислення:

1 . Обчислити $(x_0,u_0)...(x_{k+m},u_{k+m})$ за формулою $u_j = f(x_j)$.

2 . Обчислити $(x_0,v_0)...(x_{k+m},v_{k+m})$ за формулою $v_j = g(x_j)$.

3 . Обчислити $(x_0,y_0)...(x_{k+m},y_{k+m})$ за формулою $y_j = u_j \cdot v_j$.

4 . Відновити $h(x) = f(x)g(x)$ по набору $(x_0,y_0)...(x_{k+m},y_{k+m})$.

Обчислення п.3 виконуються за час $T_3(k,m) = O(k+m)$. Отже, складність алгоритму визначається складністю алгоритму пп 1,2 і складністю алгоритму п.4 .

Обчислення пп.1, 2 можна представити у вигляді множення матриці на вектор: Для многочлена $f(x) = a_0x^{n-1} + a_1x_{n-2}... + a_{n-1}$ покладемо

$$a = (a_0, a_1,..., a_{n-1}),\ y = (y_0, y_1,..., y_{n-1}),\ A = [x_j^k]_{j,k=0}^{n-1,n-1} \tag{5.4}$$

тоді

$$y = Aa \tag{5.5}$$

У розгорнутому вигляді (5.5) виглядає наступним чином:

$$\begin{bmatrix} y_0 \\ y_1 \\ ... \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} x_0^{n-1} & x_0^{n-2} & ... & 1 \\ x_1^{n-1} & x_1^{n-2} & ... & 1 \\ . & . & ... & . \\ x_{n-1}^{n-1} & x_{n-1}^{n-2} & ... & 1 \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ ... \\ a_{n-1} \end{bmatrix}$$

З курсу алгебри [7, 8] відомо, що $n \times n$ матриця невироджена. Її визначник називається визначником Вандермонда і $\det(A) = \prod_{j<k}(x_j - x_k)$. Отже,

$$a = A^{-1}y \tag{5.6}$$

Формула (5.6) задає метод п.4 - алгоритм відновлення полінома .

Евріка методу *швидкого перетворення Фур'є* полягає у тому, що в якості набору інтерполяційних точок вибираються комплексні числа – корені степеня

n з одиниці. При такому виборі перетворення (5.5) називається *дискретним перетворенням Фур'є* (DFTn) . Формула (5.5) записується як $y = DFT_n(a)$. Як ми побачимо, зворотне до (5.5) перетворення (5.6) також легко зводиться до DFT. Метод, описаний нижче, виконує пп.1, 2, 4 за час $O(n\log_2 n)$. На цей метод посилаються як на *метод швидкого перетворення Фур'є* (FFT) . Таким чином, арифметична складність алгоритму множення поліномів за часом оцінюється величиною $T(n) = O(n\log_2 n)$.

Дискретне перетворення Фур'є. З курсу алгебри [7, 8] відомо, що рівняння $x^n = 1$ має n різних комплексних коренів $\varepsilon_n^{(1)},...,\varepsilon_n^{(n)}$, причому

$$\varepsilon_n^{(j)} = \cos(2j\pi/n) + i\sin(2j\pi/n) = e^{i(2j\pi/n)}, \quad j = 1,2...,n.$$

Алгебраїчне число $\varepsilon_n^{(1)} = \cos(2\pi/n) + i\sin(2\pi/n) = e^{i(2j\pi/n)}$ називається головним значенням кореня степеня n з 1.

Якщо покласти $\varepsilon_n = \varepsilon_n^{(1)}$, то

1 . $\varepsilon_n^{(j)} = \varepsilon_n^j$,

2 . $\varepsilon_{dn}^{(dj)} = \varepsilon_n^{(j)} = \varepsilon_n^j$.

Нехай $n = 2m$. Уявімо поліном $f(x) = a_o x^{n-1} + a_1 x^{n-2} + ... + a_{n-1}$ у вигляді

$$f(x) = [a_0 x^{2m-2} + ... + a_{2m-2}]x + [a_1 x^{2m-2} + ... + a_{2m-1}]$$

Або

$$f(x) = f_L(x^2)x + f_R(x^2) \ ,$$

$$f_L(y) = a_0 y^{m-1} + ... + a_{2m-2}, \quad f_R(y) = a_1 y^{m-1} + ... + a_{2m-1} \qquad (5.7)$$

Покладемо $x = \varepsilon_{2m}^j$. Тоді $x^2 = \varepsilon_{2m}^{2j}$, $y = \varepsilon_m^j$. Оскільки $\varepsilon_{2m}^{2j} = \varepsilon_m^j$, значення набору $(f(\varepsilon_{2m}), f(\varepsilon_{2m}^2),..., f(\varepsilon_{2m}^j),...., f(\varepsilon_{2m}^{2m}))$ за наборами $(f_L(\varepsilon_m), f_L(\varepsilon_m^2),..., f_L(\varepsilon_m^j),...., f_L(\varepsilon_m^m))$, $(f_R(\varepsilon_m), f_R(\varepsilon_m^2),..., f_R(\varepsilon_m^j),...., f_R(\varepsilon_m^m))$ можна обчислити за формулою (5.7) за $4m$ арифметичних операцій ($2m$ множень і $2m$ додавань) з комплексними числами. Таким чином, арифметична складність $T(n)$ алгоритму *DFTn* над полем комплексних чисел задовольняє співвідношенню

$$T_{DFT_n}(n) = 2T_{GFT_n}(n/2) + Cn \ , \ T_{DFT_n}(2) = C,$$

звідки

$$T_{DFT_n}(n) = O(n\log_2 n).$$

Зауважимо, що $\varepsilon_{2m}^{j+m} = \varepsilon_{2m}^m \varepsilon_{2m}^j = -\varepsilon_{2m}^j$. Тому обчислення значень $f(\varepsilon_{2m}^j)$ при $j \geq m$ можна здійснювати за формулою $f(\varepsilon_{2m}^j) = -f_L(\varepsilon_m^{j-m})\varepsilon_m^{j-m} + f_R(\varepsilon_m^{j-m})$. Це

оптимізуюче перетворення зменшує мультиплікативну константу алгоритму *DFTn*. Нехай `F, FL, FR` – масиви значень функцій f, f_L, f_R. Тоді обчислення виконуються в циклі

```
m := n div 2;
for k = 0 to  m-1 do begin
  G := x*F_R[k];
  F[k] := F_L[k] + G;
  F[k+m] := F_L[k] - G,
  x := x*ε_n
end;
```

Завдання п.4 відновлення коефіцієнтів полінома $h(x) = f(x)g(x)$ вирішується *зворотним перетворенням Фур'є*. Обчислимо для цього матрицю Φ підставивши в A значення $x_j = \varepsilon_n^j$. Отримаємо $\Phi = [\varepsilon_n^{jk}]_{j,k=0}^{n-1,ni1}$. Покажемо, що для матриці Φ зворотною є матриця $\Phi^{-1} = \frac{1}{n}[\varepsilon_n^{-kj}]_{k,j=0}^{n-1,n-1}$. Для цього достатньо перемножити Φ на Φ^{-1}.

$$\Phi\Phi^{-1} = \begin{bmatrix} \varepsilon_n^{n-1} & \varepsilon_n^{n-2} & \dots & 1 \\ \varepsilon_n^{2(n-1)} & \varepsilon_n^{2(n-2)} & \dots & 1 \\ . & . & \dots & . \\ \varepsilon_n^{(n-1)(n-1)} & \varepsilon_n^{(n-1)(n-2)} & \dots & 1 \end{bmatrix} \begin{bmatrix} \varepsilon_n^{-(n-1)} & \varepsilon_n^{-2(n-1)} & \dots & \varepsilon_n^{-(n-1)(n-1)} \\ \varepsilon_n^{-(n-2)} & \varepsilon_n^{-2(n-2)} & \dots & \varepsilon_n^{-(n-1)(n-2)} \\ . & . & \dots & . \\ 1 & 1 & \dots & 1 \end{bmatrix} = \begin{bmatrix} n & 0 & \dots & 0 \\ 0 & n & \dots & 0 \\ . & . & . & . \\ 0 & 0 & \dots & n \end{bmatrix}$$

Оскільки

$$(\varepsilon_n^{j(n-1)} \quad \varepsilon_n^{j(n-2)} \quad \dots \quad 1) \cdot (\varepsilon_n^{-k(n-1)} \quad \varepsilon_n^{-k(n-2)} \quad \dots \quad 1)^T = \sum_{l=n-1}^{0} \varepsilon_n^{(j-k)l} = 0 \text{ при, } j \neq k$$

$$(\varepsilon_n^{j(n-1)} \quad \varepsilon_n^{j(n-2)} \quad \dots \quad 1) \cdot (\varepsilon_n^{-j(n-1)} \quad \varepsilon_n^{-j(n-2)} \quad \dots \quad 1)^T = \sum_{l=n-1}^{0} \varepsilon_n^{0} = n .$$

Залишилося вказати, що матриця $[\varepsilon_n^{-kj}]_{k,j=0}^{n-1,n-1}$ дорівнює матриці $[\varepsilon_n^{(n-k)j}]_{k,j=0}^{n-1,n-1}$, оскільки $\varepsilon_n^{(n-k)j} = \varepsilon_n^{nj} \cdot \varepsilon_n^{-kj} = (\varepsilon_n^n)^j \cdot \varepsilon_n^{-kj} = 1^j \cdot \varepsilon_n^{-kj} = \varepsilon_n^{-kj}$, тобто відрізняється від матриці $\Phi = [\varepsilon_n^{kj}]_{k,j=0}^{n-1,n-1}$ тільки зворотного нумерацією рядків.

Метод *FFT* вимагає виконання умов :

1. Можливість переходу від $2m$ до m на будь-якому етапі обчислень.

2 . Обчислення з комплексними числами виконуються за час $O(1)$.

Перша умова виконується, якщо степені поліномів, що перемножуються – це числа виду $2^k - 1$ (див. п.5.1.1 метод Карацуби). Друга умова забезпечується не так просто. Корені з одиниці – ірраціональні числа, і точно представити їх так, щоб виконувалася ця умова, неможливо. Вихід полягає в тому, щоб перейти до обчислень в такому скінченному полі Z_p, в якому:

1. Рівняння $x^n = 1$ розкладається на лінійні співмножники, тобто має n коренів.

2 . Обчислення приводять до точних результатів.

1. Нехай $n = 2^k$. Розглянемо послідовність простих чисел виду $p_d = dn + 1$, $d = 1,2,...$ (за теоремою Діріхле, серед чисел виду нескінченно багато простих). Оскільки мультиплікативна група скінченного поля є циклічною, існує таке число $\omega \in Z_p$, що

$$Z_p = \{0, \omega, \omega^2, ..., \omega^{p-2}, \omega^{p-1}\} \ , \ \omega^{p-1} = 1.$$

Покладемо $p = p_d$, $\varepsilon = \omega^d$. Тоді $\varepsilon, \varepsilon^2, ..., \varepsilon^{n-1}, \varepsilon^n$ – корені рівняння $x^n = 1$. Можна показати, що всі вони різні. Отже, будь-яке з простих чисел послідовності $p_d = dn + 1$ задовольняє умові 1.

2. Якщо коефіцієнти поліномів f, g, що перемножуються між собою, обмежені константою M, тобто $|a_j| < M, |b_k| < M$, то коефіцієнти їх добутку $h = fg$ обмежені величиною nM^2. Тому при $p_d > nM^2$ результат обчислень буде вірним.

5.2. Факторизація поліномів

5.2.1 Факторизація поліномів над скінченним полем

Нехай $q = p^n$ та F_q – розширення простого поля F_p характеристики p. Задача факторизації полягає у факторизації нормованого вільного від квадратів полінома $g(x) \in F_q[x]$, де F_q скінченне поле характеристики p, яке є розширенням ступеня q простого поля F_0, тобто обчисленні поліномів-факторів $g_1(x), g_2(x), ..., g_m(x)$ таких, що $g(x) = g_1(x)g_2(x)...g_m(x)$ та $g_i(x) \in F_q[x]$ нормовані і попарно взаємно прості.

Попередні методи факторизації над скінченним полем. Метод розкладання на множники поліномів над скінченним полем [69] використовує підалгоритми, наведені нижче.

Звільнення від квадратів. Розкладання на множники довільного полінома $f(x) \in F_q[x]$ доцільно починати з побудови полінома без кратних коренів, який має ту саму множину коренів, що і $f(x)$. Будемо вважати поліном $f(x)$ нормованим, тобто $a_0 = 1$. Тоді $f(x)$ над полем F_0 можна представити у вигляді добутку ступенів незвідних поліномів: $f(x) = g_1^{k_1}(x)g_2^{k_1}(x)...g_m^{k_m}(x)$, де k_i – натуральні числа, а $g_i(x) \in F_0[x]$ – незвідні і також нормовані. Тоді для $f(x)$ шуканим є поліном виду $f_0(x) = g_1(x)g_2(x)...g_m(x)$.

Нехай $f'(x)$ – формальна похідна полінома $f(x)$. Тоді $r(x)=\gcd(f(x),f'(x))$ має вигляд $c\cdot g_1^{k_1-1}(x)g_2^{k_2-1}(x)...g_m^{k_m-1}(x)$. Дійсно, опустимо означення залежності (x) поліномів від x. Тоді

$$f=g_1^{k_1}g_2^{k_1}...g_m^{k_m}$$

$$f'(x)=g_1'g_1^{k_1-1}g_2^{k_2}...g_m^{k_m}+g_2'g_1^{k_1}g_2^{k_2-1}...g_m^{k_m}+...+g_m'g_1^{k_1}g_2^{k_2}...g_m^{k_m-1}=h(g_1^{k_1-1}g_2^{k_2-1}...g_m^{k_m-1})$$

де $h=g_1'g_2...g_m+g_1g_2'...g_m+...+g_1g_2...g_m'$

Оскільки $g_i(x)$ незвідні, $\gcd(g_i,h)=\gcd(g_i,g_i')=1$. Отже, $\gcd(f,f')=g_1^{k_1-1}g_2^{k_2-1}...g_m^{k_m-1}$.
Якщо F_0 - скінченне поле характеристики p, можуть мати місце такі випадки:

1. $\gcd(f(x)f'(x))\equiv 0$. Оскільки $f(x)\neq 0$, $f'(x)\equiv 0$. Тоді $f(x)=g^{p^s}(x)$, де $g(x)$ незвідний поліном. У цьому випадку $f_0(x)=g(x)$.

2. $r(x)=\gcd(f(x),f'(x))\neq 0$. Обчислюємо $f_0(x)=f(x)\ div\ r(x)$. Процедуру обчислення $f_0(x)$ називають звільненням полінома $f(x)$ від квадратів. Припустимо, що $g(x)$ вільний від квадратів, нормований і $g(x)=f(x)h(x)$, причому $f(x)$ й $h(x)$ не мають спільних коренів. Тоді $f(x)$ й $h(x)$ взаємно прості. Отже, у розкладі $f_0(x)=g_1(x)g_2(x)...g_m(x)$ $g_i(x)$ нормовані і попарно взаємно прості.

Факторизація полінома на блоки факторів рівних ступенів. Нехай $g(x)\in F_q[x]$ – вільний від квадратів нормований поліном, який треба розкласти на незвідні фактори (множники). Припустимо, що $f(x)$ – незвідний фактор $g(x)$ ступеня s, тобто $g(x)=f(x)h(x)$, причому поліноми $f(x),h(x)$ взаємно прості. Нагадаємо, що автоморфізм φ поля F_q, $q=p^m$, визначається формулою $\varphi(x)\overset{df}{=}x^q$. Розглянемо поліном $r(x)=(\varphi^s(x)-x)(\mathrm{mod}\ g(x))$. За означенням операції mod, $x^{qs}-x=u(x)g(x)+r(x)$ або

$$x^{qs}-x=u(x)\,f(x)h(x)+r(x) \qquad (5.8)$$

Але $x^{qs}-x=x(x-\beta_1)...(x-\beta_{qs-1})$, де $F_{qs}=\{0,\beta_1,...,\beta_{qs-1}\}$ – розширення поля F_q коренями полінома $F(x)$. Нехай $\alpha\in F_{qs}$ – корінь $f(x)$. Тоді, підставивши значення $x=\alpha$ у (5.8), отримаємо $\alpha^{qs}-\alpha=0, u(\alpha)\,f(\alpha)h(\alpha)=0,$ отже, $r(\alpha)=0$. Тому $f(x)$ – спільний дільник поліномів $r(x)$ і $g(x)$. Оскільки цей факт має місце для всіх дільників фіксованого ступеня s полінома $g(x)$, найбільший спільний дільник $r(x)$ і $g(x)$ дорівнює добутку всіх дільників $g(x)$ ступеня s: $g(x)=f_1(x)...f_l(x)h(x)$, $\deg(f_j(x))=s, j=1,...,l$.

Тим самим визначено метод факторизації поліном $g(x)$ на «блоки» - фактори рівних ступенів: $g(x)=g_1(x)...g_k(x)$, де кожен з поліномів $g_j(x)$ є добутком

незвідних поліномів одного і того ж ступеня. Тепер можна вважати, що $g(x)$ є блоком: $g(x) = g_1(x)...g_k(x)$, причому ступінь факторів блоку відома й дорівнює s.

5.2.2. Метод Берлекампа

Задача полягає у факторизації нормованого вільного від квадратів полінома $g(x) \in F_q[x]$, де F_q – скінченне поле характеристики p, яке є розширенням ступеня q простого поля F_0, тобто обчисленні поліномів-факторів $g_1(x), g_2(x), ..., g_m(x)$ таких, що $g(x) = g_1(x)g_2(x)...g_m(x)$ та $g_i(x) \in F_q[x]$ нормовані і попарно взаємно прості.

Лема

Нехай $g(x) \in F_q[x]$ і $h(x) \in F_q[x]$ нормалізований поліном такий, що $h^q \equiv h(\mathrm{mod}\, g)$. Тоді

$$g(x) = \prod_{c \in F_q} \gcd(g(x), h(x) - c). \qquad (5.9)$$

Доведення

1. Насправді, права частина рівності ділить $g(x)$, оскільки $\gcd(g,h)$ ділить g для будь-якого полінома h.
2. З іншого боку, $h^q \equiv h(\mathrm{mod}\, g)$ означає, що $h^q(x) - h(x) = s(x)g(x)$. Розкладемо $h^q - h$ на множники:

$$h^q - h = h(h^{q-1} - 1) = h(h - \varepsilon_1)(h - \varepsilon_2)...(h - \varepsilon_{q-1}),$$

де ε_j, $j \in 1..q-1$ – корені ступеня $q-1$ над полем F_q. Множина $\{\varepsilon_1,...,\varepsilon_{q-1}\}$ є множиною усіх ненульових елементів поля F_q, тобто $F_q = \{0, \varepsilon_1,...,\varepsilon_{q-1}\}$. Отже,

$$s(x)g(x) = h(h - \varepsilon_1)(h - \varepsilon_2)...(h - \varepsilon_{q-1}),$$

звідки слідує, що $g(x)$ розкладається на множники виду $\gcd(g(x), h(x) - c), c \in F_q$. Таким чином, $g(x)$ ділить праву частину (5.9). Оскільки поліноми обох частин рівності (5.9) ділять один одного й нормалізовані, має місце (5.9). *Лему доведено.*

Права частина (5.9) є розкладом $g(x)$ на множники виду $\gcd(g, h(x) - c)$. Отже, потрібно побудувати поліном $h(x) \in F_q[x]$ такий, що $h^q \equiv h(\mathrm{mod}\, g)$. Зауважимо, що якщо $\deg(h) \geq \deg(g)$, формула (5.9) може дати множник $g(x)$. Тому будемо шукати такі поліноми $h(x)$, $\deg(h) < \deg(g)$. Розглянемо порівняння

$$h^q \equiv h(\mathrm{mod}\, g),\ \deg(h) < \deg(g). \qquad (5.10)$$

Кожний розв'язок цього порівняння має задовольняти системі порівнянь $h(x) \equiv c_j \pmod{g(x)}, j = 1,2,...,k$ для деякого набору $(c_1,...,c_j,...c_k)$ елементів основного поля. Тоді

$$g = g_1 g_2 ... g_k,\ g_j(x) = \gcd(g(x), h(x) - c_j) \tag{5.11}$$

Нехай

$$n = \deg(g),\ h(x) = b_0 + b_1 x + ... + b_{n-1} x^{n-1}.$$

де $b_0, b_1, ..., b_{n-1}$ – невідомі. Порівняння (5.10) перепишемо у розгорнутому виді:

$$(b_0 + b_1 x + ... + b_{n-1} x^{n-1})^q \equiv b_0 + b_1 x + ... + b_{n-1} x^{n-1} \pmod{g(x)} \tag{5.12}$$

У довільному полі характеристики p має місце тотожність $(A+B)^p \equiv A^p + B^p$, і як наслідок – тотожність $(A+B)^{p^k} \equiv A^{p^k} + B^{p^k}$. Крім того, для довільного елемента $a \in F_q$ виконується рівність $a^q = a$. Враховуючи ці рівності, (5.12) спрощується:

$$b_0 + b_1 x^q + ... + b_{n-1} x^{(n-1)q} \equiv b_0 + b_1 x + ... + b_{n-1} x^{n-1} \pmod{g(x)} \tag{5.13}$$

Отже, порівняння (5.10) зводиться до (5.13), а (5.13) зводиться до системи однорідних лінійних рівнянь над полем F_q у такий спосіб:

1. Обчислимо $x^{iq} \pmod{g(x)}$ для $i = 0,1,...,n-1$. Одержимо:

$$x^{iq} = b_{i0} + b_{i1} x + ... + b_{in-1} x^{n-1} \bmod g(x), i = 0,1,...,n-1.$$

Підставимо ці значення у (5.13). Отримаємо поліноміальне порівняння ступеня $n-1$ виду

$$B_0 + B_1 x + ... + B_{n-1} x^{n-1} \equiv b_0 + b_1 x + ... + b_{n-1} x^{n-1} \pmod{g(x)},$$

де B_j - лінійні комбінації невідомих $b_0, b_1, ..., b_{n-1}$ з коефіцієнтами з поля F_q. Але ступінь лівої частини цього порівняння менший за $n = \deg(g)$. Тому $B_0 + B_1 x + ... + B_{n-1} x^{n-1} = b_0 + b_1 x + ... + b_{n-1} x^{n-1}$, і отже,

$$B_0 - b_0 = B_1 - b_1 = ... = B_{n-1} - b_{n-1} = 0 \tag{5.14}$$

2. Через B позначимо $n \times n$ матрицю (b_{ij}) однорідної системи рівнянь над F_q. Вектор невідомих $b = (b_0, b_1, ..., b_{n-1})$ буде розв'язком порівняння (5.10) тоді й тільки тоді, коли b є розв'язком однорідної системи рівнянь (5.14). Система (5.14) має вид $u(B-E) = 0$. Вона має q^k фундаментальних розв'язків. Для кожного з цих розв’язків поліном $h(x) = b_0 + b_1 x + ... + b_{n-1} x^{n-1}$ є

розв'язком порівняння (5.10). Ця система еквівалентна системі $(B-E)^T u = 0$, яку можна розв'язувати, наприклад, методом Гаусса виключення змінних.

Зауваження. Метод Гаусса елементарними перетвореннями рядків розширеної матриці $\left[(B-E)^T, u\right]$ приводить вихідну систему спочатку до трапецієподібного виду (прямій хід), а потім будує систему фундаментальних розв'язків приведенням до діагонального виду (зворотний хід). Легко бачити, що один з фундаментальних розв'язків відповідає поліному-константі $h(x) \equiv 1$. Будь-який інший фундаментальний розв'язок відповідає поліному $h(x)$, який можна використовувати для факторизації $g(x)$ по формулах (5.9), (5.11).

Використання результанта в методі Берлекампа. З (5.9) - (5.14) видно, що для кожного з нетривіальних фундаментальних розв'язків $h_j(x)$ алгоритм вимагає пошуку таких елементів c_j поля F_q, для яких $\gcd(g(x), h(x) - c_j)$ не дорівнює 1. Очевидний алгоритм такого пошуку – повний перелік усіх елементів поля F_q як значень c_j. Покажемо, як спростити обчислення $\gcd(g(x), h(x) - c_j)$, використовуючи поняття результанта двох поліномів.

Означення 5.1. Результантом поліномів [7, 19]

$$g(x) = a_0 + a_1 x + \ldots + a_s x^s,\ h(x) = b_0 + b_1 x + \ldots + b_m x^m$$

називається визначник, складений з коефіцієнтів $g(x), h(x)$ виду

$$R_x(h(x), g(x)) = \begin{vmatrix} a_0 & a_1 & \ldots & a_s & 0 & 0 \\ . & . & . & . & . & . \\ 0 & 0 & a_0 & a_1 & \ldots & a_s \\ b_0 & b_1 & \ldots & b_m & 0 & 0 \\ . & . & . & . & . & . \\ 0 & 0 & b_0 & b_1 & \ldots & b_m \end{vmatrix} \qquad (5.15)$$

В (5.15) кількість рядків – коефіцієнтів $g(x)$ дорівнює m, а кількість рядків – коефіцієнтів $h(x) - u$ дорівнює s.

Розглянемо пару поліномів $h(x) - u,\ g(x)$, де u - змінна. Відомо [7,13, 19], що вони мають нетривіальний спільний дільник тоді й тільки тоді, коли результант $R_x(h(x) - u, g(x)) = 0$. Нехай

$$g(x) = x^s + a_1 x^{s-1} + \ldots + a_s,\quad h(x) = x^m + b_1 x^{m-1} + \ldots + b_m\ .$$

Тоді

$$R_x(h(x)-u,g(x))=\begin{vmatrix} 1 & a_1 & \dots & a_s & 0 & 0 \\ . & . & . & . & . & . \\ 0 & 0 & 1 & a_1 & \dots & a_s \\ 1 & b_1 & \dots & b_m-u & 0 & 0 \\ . & . & . & . & . & . \\ 0 & 0 & 1 & b_1 & \dots & b_m-u \end{vmatrix}.$$

Легко бачити, що $R_x(h(x)-u,g(x))$ – поліном ступеня s, який позначимо через $R_{g,h}(u)$. Обчислення полінома $R_{g,h}(u)$ можна здійснити методом інтерполяції Лагранжа, обчисливши визначник $R_x(h(x)-u,g(x))$ для $s+1$ значення c_j, а потім побудувавши інтерполяційний поліном Лагранжа по цим значенням.

Тепер перевірка $\gcd(g(x),h(x)-c_j)\neq 1$ полягає в обчисленні $R_{g,h}(c_j)=0$. Пошук усіх потрібних значень c_j можна здійснити методом Цассенхауса [68].

5.2.3. Метод факторизації поліномів над полем раціональних чисел

Ефективний алгоритм розкладання на множники поліномів з раціональними коефіцієнтами, реалізований у багатьох системах комп'ютерної алгебри, використовує метод Берлекампа.

Нехай $f(x)=a_nx^n+\dots+a_0\in Q[x]$. Приведенням коефіцієнтів до спільного знаменника $f(x)$ можна представити у вигляді $\frac{d}{c}(c_nx^n+\dots+c_0)$, де c – найменший спільний знаменник дробів $a_0,a_1,\dots,a_n$, $c_n,\dots,c_1,c_0$ - цілі числа, а d – їх найбільший спільний множник. Розкладання на множники чисел c,d не розглядаються.

Отже, задача зводиться до розкладання на множники полінома із цілими коефіцієнтами: $f(x)=a_nx^n+\dots+a_0\in Z[x]$. За лемою Гаусса [20, 21], усі дільники $f(x)$ – поліноми із цілими коефіцієнтами. Нехай $g(x)=b_mx^m+\dots+b_0\in Z[x]$ - дільник полінома $f(x)$.

Виберемо непарне просте число p й покладемо $r=p/2$. Розглянемо гомоморфізм $\varphi_p: Z\to Z_p$, де $Z_p=\{-r,-r+1,\dots,-1,0,1,\dots,r-1,r\}$, визначений рівностями

$$\varphi_p(z)=\begin{cases} |z| \bmod p, & \text{если } |z| \bmod p\le r, \\ -|z| \bmod p, & \text{если } |z| \bmod p> r. \end{cases}$$

Через $f_{(p)}(x)$, $g_{(p)}(x)$ позначимо поліноми $\varphi(a_n)x^n+\dots+\varphi(a_0)$, $\varphi(b_m)x^m+\dots+\varphi(b_0)$. Тоді, якщо над кільцем Z $f(x)=g(x)h(x)$, над полем G_p має місце факторизація $f_{(p)}(x)=g_{(p)}(x)h_{(p)}(x)$. Ідея методу полягає у тому, щоб використовувати розклад $f_{(p)}(x)$ над полем G_p для факторизації $f(x)$ над кільцем Z.

Перший варіант реалізації цієї ідеї полягає в тому, щоб обрати настільки велике число p, що матиме місце рівність $f_{(p)}(x) \equiv f(x)$, і кожний з факторів $g_{(p)}(x) \equiv g(x)$, тобто $\varphi_p(a_i) = a_i$ й $\varphi_p(b_j) = b_j$. Тоді будь-який дільник $g(x)$ полінома $f(x)$ над Z дорівнюватиме деякому дільнику $f(x)$ над G_p. Оцінка простого числа p, для якого $\varphi_p(a_i) = a_i$ й $\varphi_p(b_j) = b_j$ дають нерівності, встановлені в наступних теоремах.

Для довільного полінома $h(x) = c_k x^k + ... + c_0$ через $\|h\|$ позначимо число $\|h(x)\| = \sqrt{c_0^2 + ... + c_k^2}$. Нехай поліном з цілими коефіцієнтами $g(x) = b_m x^m + b_{m-1} x^{m-1} + ... + b_0$ – дільник полінома з цілими коефіцієнтами $f(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_0$. Тоді мають місце нерівності:

Нерівність Ландау-Миньотта [20]:

$$\sum_{j=0}^{m} |b_j| \le 2^n \left| \frac{a_n}{b_m} \right| \|f\| \tag{5.16}$$

Нерівність Миньотта [72]:

$$|b_j| \le C_{n-1}^{j} \|f\| + C_{n-1}^{j-1} |a_n| \tag{5.17}$$

$$\|g\| \le \sqrt{C_m^{2m}} \|f\| \tag{5.18}$$

$$|b_j| \le C_{m/2}^{m-1} (\|f\| + \max(|a_0|,...,|a_n|) \tag{5.19}$$

Нерівності (5.16) - (5.19) можна використовувати для пошуку простого числа p. Нехай, наприклад, $B = 2C_{m/2}^{m-1} (\|f\| + \max(|a_0|,...,|a_n|)$. Тоді в якості p слід обрати найменше просте число, більше за B.

Власно алгоритм факторизації полінома $f(x) \in Z[x]$ полягає у випробуваннях усіх можливих дільників цього полінома над полем G_p. Для кожного з дільників $g_j(x)$ перевіряється подільність $f(x)$ на $g_j(x)$ над Z.

Зауваження. Той факт, що $g(x)$ – дільник $f(x)$ над скінченним полем G_p, ще не означає, що $g(x)$ – дільник $f(x)$ над кільцем Z. Існують незвідні над Z поліноми, які розкладаються на множники над будь-яким скінченним полем [20].

Недоліком цього підходу є велика кількість випробувань елементів поля G_p. Тому розглянемо інший варіант. Суть його полягає в тому, щоб у рамках модулярного підходу розкладати на множники $f_{(p)}(x)$ для невеликих значень p, а потім застосувати один з так званих методів підйому, що приводять до збільшення значення p. Наведемо один такий метод.

Припустимо, що p й q – пара (невеликих) простих чисел і

$$f(x)=g(x)h(x),\ f_{(p)}(x)=g_{(p)}(x)h_{(p)}(x),\ f_{(q)}(x)=g_{(q)}(x)h_{(q)}(x),$$
$$\varphi_{(p)}(g(x))=g_{(p)}(x),\ \varphi_{(q)}(g(x))=g_{(q)}(x).$$

Тепер треба обчислити поліном $g_{(pq)}(x)$ по поліномах $g_{(p)}(x), g_q(x)$, застосовуючи до пар відповідних коефіцієнтів алгоритми китайської теореми про остачі, п.1.1.4.

Ще один метод підйому використовує так звану лему Гензеля [20]. Нехай $f_{(p)}(x)=g_{(p)}(x)h_{(p)}(x)$ над G_p, поліном $f_{(p)}(x)$ вільний від квадратів і нормований, а $g_{(p)}(x), h_{(p)}(x)$ взаємно прості. Наша мета полягає в тому, щоб обчислити розклад $f_{(p^k)}(x)=g_{(p^k)}(x)h_{(p^k)}(x)$ для довільно великого значення k. Для $k=2$ маємо $f_{(p^2)}(x)=g_{(p^2)}(x)h_{(p^2)}(x)$. Це означає, що над кільцем Z мають місце порівняння

$$(f_{(p)}(x)=f_{(p^2)})(x)\bmod(p),(g_{(p)}(x)=g_{(p^2)}(x))\bmod(p),\ (h_{(p)}(x)=h_{(p^2)}(x))\bmod(p).$$

Тому, опускаючи у формулах позначення змінної, маємо:

$$f_{(p^2)}=f_{(p)}+pf_0, g_{(p^2)}=g_{(p)}+pg_0, h_{(p^2)}=h_{(p)}+ph_0, \tag{5.20}$$

$$f_{(p)}+pf_0=(g_{(p)}+pg_0)(h_{(p)}+ph_0)\bmod(p^2), \tag{5.21}$$

де f_0, g_0, h_0 – деякі поліноми. Обчисливши їх за формулами (5.20), можна «підняти» $f_{(p)}, g_{(p)}, h_{(p)}$ до $f_{(p^2)}, g_{(p^2)}, h_{(p^2)}$.

Оскільки $f_{(p)}=g_{(p)}h_{(p)}\bmod(p)$, розкриваючи дужки й спрощуючи (5.21), одержимо

$$\left(\frac{f_{(p)}-g_{(p)}h_{(p)}}{p}+f_0=g_0h_{(p)}+g_{(p)}h_0\right)\bmod(p) \tag{5.22}$$

Права частина (5.22) містить невідомі – поліноми g_0, h_0, а всі члени лівої частини відомі. Оскільки поліноми $f_{(p)}, g_{(p)}, h_{(p)}$ нормовані, ступінь лівої частини не перевершує n. Для того, щоб отримати праву частину, застосуємо розширений алгоритм Евкліда до поліномів $g_{(p)}, h_{(p)}$. Оскільки $g_{(p)}, h_{(p)}$ взаємно прості, розширений алгоритм Евкліда обчислить такі g_1, h_1, що

$$h_1g_{(p)}+g_1h_{(p)}=1. \tag{5.23}$$

Позначимо ліву частину (5.19) через L. Домножимо обидві частини (5.23) на L. Отримаємо:

$$(Lh_1)g_{(p)}+(Lg_1)h_{(p)}=L. \tag{5.24}$$

Поліном Lh_1 розділимо з остачею на $h_{(p)}$: $Lh_1 = lh_{(p)} + h_1^{'}$. Аналогічно, $Lg_1 = tg_{(p)} + g_1^{'}$. Підставимо значення Lh_1, Lg_1 в (5.24). Одержимо: $(lh_{(p)} + h_1^{'})g_{(p)} + (tg_{(p)} + g_1^{'})h_{(p)} = L$. Розкриваючи дужки й групуючи подібні, отримаємо:

$$(l+t)h_{(p)}g_{(p)} + h_1^{'}g_{(p)} + g_1^{'}h_{(p)} = L \qquad (5.25)$$

Порівнюючи ступені лівої та правої частин, отримаємо, що $l + t = 0$ і $h_1^{'}g_{(p)} + g_1^{'}h_{(p)} = L$. Таким чином, $h_1^{'}$, $g_1^{'}$ – шукані розв'язки (5.22): $h_0 = h_1^{'}$, $g_0 = g_1^{'}$.

Аналогічно підйому $p \to p^2$ здійснюється підйом $p^2 \to p^3$, і так далі, від $p^{k-1} \to p^k$. При підйомі $p^2 \to p^3$ формули (5.20), (5.22) мають вигляд:

$$f_{(p^3)} = f_{(p^2)} + pf_0,\ g_{(p^3)} = g_{(p^2)} + pg_0,\ h_{(p^3)} = h_{(p^2)} + ph_0,$$

$$(\frac{f_{(p^2)} - g_{(p^2)}h_{(p^2)}}{p^2} + f_0 = g_0h_{(p^2)} + g_{(p^2)}h_0) \bmod(p),$$

а в загальному випадку –

$$f_{(p^k)} = f_{(p^{k-1})} + pf_0,\ g_{(p^k)} = g_{(p^{k-1})} + pg_0,\ h_{(p^k)} = h_{(p^{k-1})} + ph_0,$$

$$(\frac{f_{(p^{k-1})} - g_{(p^{k-1})}h_{(p^{k-1})}}{p^{k-1}} + f_0 = g_0h_{(p^{k-1})} + g_{(p^{k-1})}h_0) \bmod(p).$$

5.3. Конструктивні методи теорії поліноміальних ідеалів

5.3.1. Базиси Гребнера

Класична теорія поліноміальних ідеалів досліджує структуру та алгебраїчні властивості систем алгебраїчних рівнянь, що розглядаються разом з усіма своїми наслідками. Конструктивні методи теорії поліноміальних ідеалів знайшли ефективну реалізацію в теорії базисів Гребнера [71]. Нижче наведені основні відомості з теорії базисів Гребнера, які є, як ми покажемо, ефективним інструментом розв'язання основних задач конструктивної теорії поліноміальних ідеалів. Викладення слідує [71, 73].

Означення 5.2. Нехай R – комутативне кільце. Множина $I \subseteq R$ називається ідеалом кільця R, якщо виконуються властивості: Для будь яких f, g

1. $f, g \in I \to f - g \in R$;
2. $f \in R, g \in I \to f \cdot g \in I$.

Означення 5.3. Нехай F – деяке поле, $X = (x_1, x_2, ..., x_n)$ – вектор змінних, $F[x_1, x_2, ..., x_n]$ – кільце поліномів від n змінних $x_1, x_2, ..., x_n$ над F. Кільце

$F[x_1, x_2, ..., x_n]$ позначатимемо також через $F[X]$. Нехай, далі $J = \{f_1, f_2, ..., f_m\}$ – скінченна множина поліномів з $F[X]$. Розглянемо множину поліномів

$$I = \{ f \in F[X] \mid f = g_1 f_1 + g_2 f_2 + ... + g_m f_m \ \forall g_1, ..., g_m \in F[X] \}$$

Отже, множина I є множиною усіх лінійних комбінацій поліномів із $I = \{f_1, f_2, ..., f_m\}$ з поліноміальними коефіцієнтами $g_i \in F[X]$.

Множина I задовольняє означенню 5.2, отже, I є ідеалом кільця $F[X]$, $J \subset I$, а також I є найменшим ідеалом, що містить J. Цей ідеал позначатимемо через $Ideal(J)$, або просто (J). Кажуть, що множина J породжує I, а також, що J є базисом I.

Крім того, будемо писати $f \equiv g (\mathrm{mod}\, I)$, якщо f порівняне з g за модулем $Ideal(I)$, тобто $f - g \in Ideal(I)$.

Розглянемо систему алгебраїчних рівнянь

$$f_1(X) = 0 \,\&\, f_2(x) = 0 \,\&\, ... \,\&\, f_m(X) = 0 \qquad \textbf{(5.26)}$$

Очевидно, що $g_1 f_1 + g_2 f_2 + ... + g_m f_m = 0$ є наслідком цієї системи, тобто

$$f_1(X) = 0 \,\&\, f_2(x) = 0 \,\&\, ... \,\&\, f_m(X) = 0 \to g_1 f_1 + g_2 f_2 + ... + g_m f_m = 0.$$

Очевидно також, що співвідношення означення **5.2** можна переписати у вигляді

$$f(X) \,\&\, g(X) = 0 \to f(X) - g(X) = 0, \quad g(X) = 0 \to f(X) \cdot g(X) = 0.$$

Отже, множина лівих частин наслідків системи (5.26) утворює ідеал $Ideal(J)$, породжений набором лівих частин системи (5.26).

Означення 5.4. Скінченна множина поліномів $J = \{f_1, f_2, ..., f_m\}$ є базисом ідеалу $I \subset F[X]$, якщо $I = Ideal(J)$.

Має місце теорема Д. Гільберта про базис:

Теорема 5.1. Будь-який ідеал I кільця поліномів $F[X]$ має скінченний базис.

Нехай $X = \{x_1, ..., x_n\}$ – множина змінних і $K = (k_1, ..., k_n)$ – вектор натуральних чисел. Вираз $x_1^{k_1}, ..., x_n^{k_n}$ будемо називати степенем і позначати через X^K.

Вираз $a \cdot X^K, a \in F$ будемо називати мономом. За означенням, поліном є скінченною сумою мономів:

$$f(X) = M_1 + M_2 + ... + M_l = a_1 D_1 + a_2 D_2 + ... + a_l D_l = a_1 X^{K_1} + a_2 X^{K_2} + ... + a_l X^{K_l} \qquad (5.27)$$

Представлення полінома у вигляді суми мономів є однозначним, якщо на множині степенів установити деякий фіксований лінійний порядок. Звичайно використовують або лінійний порядок, індукований ступенем, або чистий лексикографічний порядок.

За ступеневим порядком, $X^{K_1} \prec X^{K_2}$, якщо

$$\sum_{j=1}^{n} k_{1j} < \sum_{j=1}^{n} k_{2j} \text{ або } \sum_{j=1}^{n} k_{1j} = \sum_{j=1}^{n} k_{2j},\ k_{11} = k_{21},...k_{1l-1} = k_{2l-1}, k_{1l} < k_{2l}.$$

За чисто лексикографічним порядком, $X^{K_1} \prec X^{K_2}$, якщо

$$k_{11} = k_{21},...k_{1l-1} = k_{2l-1}, k_{1l} < k_{2l}.$$

Лінійне упорядкування мономів породжує лінійне упорядкування мономів з точністю до коефіцієнтів.

Визначені лінійні упорядкування задовольняють наступним умовам:

1. $1 \prec M$ для будь–якого монома $M \neq 1$;
2. якщо $M_1 \prec M_2,$, то $M_1 \cdot M \prec M_2 \cdot M$.

Лінійний порядок, що задовольняє (1, 2) будемо називати допустимим. Зафіксуємо деякий допустимий порядок «$\prec$». Відносно цього порядку введемо наступні позначення. Якщо у (5.27)

$$M_1 \succ M_2 \succ ... \succ M_l \quad (D_1 \succ D_2 \succ ... \succ D_l) \text{ та } a_1 \neq 0$$

$Coef(f, M)$ – коефіцієнт при мономі M в f;

$LeadMon(f) = M_1$ – старший моном;

$LeadCoef(f) = a_1$ – коефіцієнт при старшому мономі.

Поліном g можна редукувати за допомогою полінома f, коефіцієнта b та ступеня D (позначається $g \to_{f,b,D}$), якщо мають місце рівності

$$g = M_{11} + ... + M_{1j} + ... + M_{1l},\ \ M_{1j} = a_{1j} D \cdot D_{21},\ \ f = a_{21} D_{21} + ... + M_{2k},\ \ b = a_{1j} / a_{21}.$$

Означення 5.5. Поліном h є редукцією полінома g за допомогою полінома f, якщо $g \to_{f,b,D}$ і $h = g - b \cdot D \cdot f$.

Нехай $I = \{f_1,..., f_k\}$. Поліном h є редукцією полінома g за модулем ідеалу (I) (позначається $g \to_I h$), якщо знайдуться такі $f \in I, b, D$, що $g \to_{f,b,D}$, і h є редукцією полінома g за допомогою полінома f.

Інакше кажучи, h є редукцією g за модулем ідеалу I, якщо h отримано із g відніманням підходящого добутку $b \cdot D \cdot f$, і при цьому старший моном полінома $b \cdot D \cdot f$ збігається з деяким мономом полінома g.

Означення 5.6. Поліном h представлено у нормальній формі (або скороченій формі) за модулем I, якщо не існує такого полінома h', що $h \to_F h'$.

Поліном h є нормальною формою полінома g за модулем ідеалу (I) (позначається $h = NF(I,g)$), якщо існує така послідовність редукцій $g = h_0 \to_I h_1 \to_I h_2 \to_I ... \to_I h_m = h$, що h представлений у нормальній формі за модулем I.

Наступний алгоритм є алгоритмом приведення полінома g до нормальної форми за модулем ідеалу (I).

Алгоритм 5.1 $\{h = NF(I,g)\}$

$h := g$;

While існують такі $f \in I,b,D$, що $h \to_{f,b,D}$

do(

обрати такі $f \in F,b,D$ із найстаршим мономом $D \cdot LeadMon(f)$,

що $h \to_{f,b,D}$;

$h := h - b \cdot D \cdot f$

);

Для коректності алгоритму вибір найстаршого монома не є необхідним. Цей вибір здійснюється для ефективності алгоритму. Закінчення роботи алгоритму гарантується наступною властивістю *нетеровості* процесу редукування.

Для кожного I відношення $\to_I$ є *нетеровим*. Це означає, що будь-яка послідовність редукцій $h_0 \to_I h_1 \to_F h_2 \to_I ...$ є скінченною.

Взагалі кажучи, нормальна форма полінома g за модулем I не є канонічною. Ті множини I, для яких форма $h = NF(I,g)$ є канонічною, відіграють ключову роль у алгоритмічному підході до розв'язання задач теорії поліноміальних ідеалів, заснованому на базисах Гребнера.

Означення 5.7. *Набір поліномів* $I = \{f_1,..., f_m\}$ називається базисом Гребнера, якщо для будь-яких поліномів g, h_1, h_2, таких, що h_1 і h_2 – нормальні форми полінома g за модулем I, $h_1 = h_2$.

Основна мета полягає в тому, щоб показати, що

1. За будь–яку множиною поліномів I можна побудувати множину J таку, що J вже є базисом Гребнера, причому $Ideal(I) = Ideal(J)$.

2. Багато важливих алгоритмічних проблем конструктивної теорії поліноміальних ідеалів, можуть бути ефективно вирішені за допомогою базисів Гребнера.

Властивість Черча-Россера. Нехай на множині термів деякої алгебраїчної теорії задане відношення $t \to r$, яке називають відношенням редукції. Через $t -* \to r$ позначимо транзитивне замикання відношення $t \to r$. Кажуть, що відношенню $\to$ притаманна властивість Черча-Россера, якщо

1. Відношенню $t \to r$ притаманна властивість конфлюентності: для будь-якого терма t, термів t_1, t_2 таких, що $t -* \to t_1, t -* \to t_2$,існує такий терм t_3, що $t_1 -* \to t_3, t_2 -* \to t_3$.
2. Відношенню $t \to r$ притаманна властивість нетеровості: будь яка послідовність $t_0 \to t_1 \to ... \to t_k \to ...$ є скінченною.

Більшість алгоритмічних застосувань базисів Гребнера засновані на наступній їхній фундаментальній властивості.

Теорема 5.2. (характеризація базисів Гребнера).

Нехай $A(g, I)$ – алгоритм побудови нормальної форми довільного полінома g за модулем ідеалу (I). Наступні умови рівносильні:

1. Множина поліномів I є базисом Гребнера.

2. Для будь–яких поліномів f, g порівняння $f \equiv g (\mathrm{mod}\, I)$ еквівалентно рівності $A(f, I) = A(g, I)$.

3. Відношенню редукції $t \to r$ притаманна властивість Черча – Россера.

5.3.2. Алгоритмічна побудова базисів Гребнера

Для перевірки того, чи є дана множина I базисом Гребнера та для побудови таких базисів важливу роль грає поняття S–полінома та операції обчислення S–полінома.

Означення 5.8. S–поліномом від поліномів f_1, f_2 називається поліном $SP(f_1, f_2) = D_1 \cdot f_1 - \frac{c_1}{c_2} \cdot D_2 \cdot f_2$, де $c_i = LeadCoef(f_i)$, степені D_i такі, що $S_1 \cdot D_1 = S_2 D_2$ і $S_1 \cdot D_i$ є найменшим спільним кратним мономів S_1, S_2, $S_i = LeadMon(f_i)$.

Відзначимо, що найменше спільне кратне мономів S_1, S_2 дорівнює мінімальному моному, що редукується як за модулем f_1, так і за модулем f_2. Алгоритмічний критерій для базисів Гребнера сформульовано у наступній теоремі.

Теорема 5.3. (алгоритмічна характеризація базису Гребнера).

Нехай $A(g, I)$ – довільний алгоритм побудови нормальної форми. Для будь-яких g, I. Наступні властивості є рівносильними:

1. I – базис Гребнера;

2. для будь–яких $f_1, f_2 \in I$ виконано $A(I, SP(f_1, f_2)) = 0$.

Умова (2) дозволяє перевіряти властивість «I – базис Гребнера»: достатньо розглянути скінченне число пар $f_1, f_2 \in I$, обчислити відповідні S–поліноми та встановити, чи редукуються вони до нуля за допомогою застосування алгоритму нормальної форми S. Крім того, теорема 5.2 є фундаментом основного алгоритму 5.2 для розв'язання наступної задачі.

Задача 5.1. Дано множину поліномів I. Знайти таку множину J, що $Ideal(I) = Ideal(J)$ і J є базисом Гребнера.

Алгоритм 5.2

$J := I$;

$B = \{(f_1, f_2) \mid f_1, f_2 \in I, f_1 \neq f_2\}$;

While $B \neq \varnothing$ **do(**

(f_1, f_2) – пари з B;

$B := B \setminus \{(f_1, f_2)\}$;

$h := SP(f_1, f_2)$;

$h' := NF(J, h)$;

if $h' \neq 0$ **then(**

$B := B \cup \{(g, h') \mid g \in J\}$;

$J := J \cup \{h'\}$.)

)

Коректність цього алгоритму частково визначається теоремою 5.2. Завершуваність його виконання можна довести, користуючись теоремою Гільберта про базис.

Означення 5.9. I називається скороченим базисом Гребнера, якщо I – базис Гребнера, і, крім того, для всіх $f \in I$ поліном f представлений у нормальній формі за модулем $I \setminus \{f\}$ й $LeadCoef(f) = 1$.

Теорема 5.4. (одиничність скороченого базису Гребнера).

Якщо $Ideal(I) = Ideal(J)$, де I та J – скорочені базиси Гребнера, то $I = J$.

Означення 5.10. Позначимо через $GrBase(I)$ функцію, що співставляє кожній множині поліномів I скорочений базис Гребнера J: $J = GrBase(I)$ якщо $Ideal(I) = Ideal(J)$.

З теорем 5.2, 5.3, алгоритму 5.2 одержуємо наступну основну теорему, у якій сформульовані алгоритмічні результати про базиси Гребнера.

Теорема 5.5. Функція $GrBase(I)$, що обчислюється алгоритмом 5.2. і має наступні властивості: для всіх I, J:

1. $Ideal(I) = Ideal(GrBase(I))$;
2. якщо $Ideal(I) = Ideal(J)$, то $GrBase(I) = GrBase(J)$;
3. $GrBase(I)$ є скороченим базисом Гребнера.

Поліпшена версія алгоритму. Наведемо наступні три можливості для прискорення обчислень.

1. Порядок вибору пар (f_1, f_2), для яких будуються S–поліноми, хоча і є логічно несуттєвим, значною мірою впливає на складність алгоритму. Як правило, пари з мінімальним щодо впорядкування $\prec$ найменшим спільним кратним старших мономів варто вибирати в першу чергу. Відповідно до наступного пункту це може значно зменшити час обчислень.

2. Щоразу, коли до базису додається черговий поліном, всі інші поліноми можна редукувати, використовуючи також і новий поліном. Тим

самим можна з J видалити декілька поліномів. Такі редукції викликають цілий каскад редукцій і видалень. Крім того, якщо ця процедура виконується систематично протягом виконання алгоритму, кінцевий результат алгоритму автоматично є скороченим базисом Гребнера. Редукція поліномів за модулем інших поліномів базису має виконуватися також і на початку виконання алгоритму.

3. Критерій, що пропонується нижче, дозволяє розпізнати для даного $S-$ поліному h, чи редукується він до нуля, без фактичного виконання редукції. Це може привести до значних скорочень обчислень. З використанням цього критерію в сприятливих ситуаціях досить розглянути S–поліноми в кількості $O(l)$, замість $O(l^2)$, де l – кількість поліномів у базисі. (Зрозуміло, у загальному випадку l змінюється у ході виконання алгоритму, і тому дуже важко врахувати теоретично вплив цього критерію.)

Лема 5.3. Нехай I – базис Гребнера відносно чисто лексикографічного порядку на мономах. Без обмеження загальності припустимо, що $x_1 \prec x_2 \prec ... \prec x_n$. Тоді $Ideal(I) \cap F[x_1, x_2, ..., x_i] = Ideal(I \cap F[x_1, x_2, ..., x_i])$ для $i = 1,2,...,n$.

Задача 5.2. Дано скінченну множину поліномів I, що має непусту скінченну множину коренів. Знайти усі корені I.

Розв'язання. Обчислимо базис Гребнера J множини I відносно чисто лексикографічного порядку мономів. Тоді змінні поліномів з J «відділені» як у лемі 5.3 (іншими словами, J має «трикутний» вид). Оскільки кільце поліномів над полем від однієї змінної є евклідовим, то J містить у точності один поліном з $F[x_1]$ (це поліном найменшого ступеня з $Ideal(J) \cap F[x_1]$).

Отже, нехай $p_1(x_1)$ – поліном з $J \cap F[x_1]$. Покладемо $X_1 = \{(a_1) \mid p(a_1) = 0\}$. Позначимо через H_1 множину поліномів з $(J \cap F[x_1, x_2]) \setminus F[x_1]$. Нехай $H_1 = \{p_{21}(x_1, x_2), ..., p_{2k}(x_1, x_2)\}$. Для кожного $a_1 \in X_1$ розглянемо множину $H_1 = \{p_{21}(x_1, x_2), ..., p_{2k}(x_1, x_2)\}$, $H_2(a_1) = \{p_{21}(a_1, x_2), ..., p_{2k}(a_1, x_2)\}$.

Покладемо $p_2(x_2) = \gcd(p_{21}(a_1, x_2), ..., p_{2k}(a_1, x_2))$. $\{p_2(x_2)\}$ є базисом Гребнера для H_1^0, оскільки для поліномів однієї змінної алгоритм побудови $GrBase(I)$ є алгоритмом Евкліда.

Нехай $X_1 = \{(a_1) \mid p(a_1) = 0\}$, $X_2(a_1) = \{(a_1, a_2) \mid p_2(a_1, a_2) = 0\}$. Покладемо $X_2 = \cup X_2(a_1)$, причому об'єднання береться по всіх множинах $X_2(a_1)$, де $a_1 \in X_1$. Відзначимо, що для деяких $a_1 \in X_1$ поліном $p_2(x_2)$ може виявитися рівним константі. У цьому випадку множина $X_2(a_1) = \varnothing$. Однак, у силу зроблених припущень $X_2 = \varnothing$.

Нехай $X_i = \{(a_1, ..., a_i)\}$ – множина коренів ідеалу $Ideal(I \cap F[x_1, ..., x_i])$. Позначимо через H_i множину поліномів з $(J \cap F[x_1, ..., x_{i+1}]) \setminus F[x_1, ..., x_i]$. Покладемо

$$H_i = \{p_{i+1,1}(x_1, ..., x_i, x_{i+1}), ..., p_{i+1,s}(x_1, ..., x_i, x_{i+1})\}.$$

Для кожного $(a_1,...,a_i) \in X_i$ розглянемо множину

$$H^0{}_i = \{p_{i+1,1}(a_1,...,a_i,x_{i+1}),..., p_{i+1,s}(a_1,...,a_i,x_{i+1})\}.$$

Нехай $p_{i+1}(x_{i+1}) = GCD(p_{i+1,1}(a_1,a_2,...,a_i,x_{i+1}),..., p_{i+1,s}(a_1,a_2,...,a_i,x_{i+1}))$. Нехай також $X_{i+1}(a_1,...,a_i) = \{(a_1,...,a_i,a_{i+1}) \mid p_{i+1}(a_{i+1}) = 0\}$. Покладемо $X_{i+1} = \cup X_{i+1}(a_1,...,a_i)$, причому об'єднання береться по всіх $X_{i+1}(a_1,...,a_i)$ таких, що $(a_1,...,a_i) \in X_i$. Продовжуючи цей процес, отримаємо множину X_n, яка є множиною коренів $Ideal(J)$.

5.3.3. Обчислення у розширеннях полів. Конструкції теорії Галуа.

У цьому пункті ми сформулюємо алгоритм побудови примітивного елементу розширення поля нульової характеристики й використаємо його для розв'язання таких задач комп'ютерної алгебри, як задача побудови групи Галуа, а значить, і задача розв'язання поліноміального рівняння в радикалах, а також інші задачі теорії Галуа (див. [17, 22]).

Означення 5.11. Нехай F – поле характеристики 0 ($ch(F) = 0$) і $f(x) \in F[x]$ – поліном ступеня n і $\alpha_1,...,\alpha_n$ – його корені. Полем розкладання $f(x) \in F[x]$ називається поле $F(\alpha_1,...,\alpha_n)$. Відома теорема теорії розширення полів – теорема про примітивний елемент – ствердження, що існує елемент $\beta \in F(\alpha_1,...,\alpha_n)$ такий, що $F(\alpha_1,...,\alpha_n) = F(\beta)$.

Використовуючи метод Кронекера [7, 8], надамо її конструктивне доведення, установивши тим самим алгоритм побудови елемента β.

Алгоритм побудови примітивного елемента (за Кронекером). Елемент β ми будемо шукати у вигляді лінійної комбінації

$$\beta = c_1 \cdot \alpha_1 + ... + c_n \cdot \alpha_n, c_i \in K, i = 1,...,n \qquad \textbf{(5.28)}$$

Розглянемо лінійну комбінацію $L = c_1 \cdot x_1 + ... + c_n \cdot x_n$, $c_i \in K$, $i = 1,...,n$ від змінних $x_1,...,x_n$ з буквеними коефіцієнтами $c_1,...,c_n$. Введемо наступні скорочення:

$$\mathrm{A} = (\alpha_1,...,\alpha_n),\ A = (a_1,...,a_n),\ C = (c_1,...,c_n),\ X = (x_1,...,x_n),\ L = L(X,C).$$

Нехай $f(x) = b_0x^n + b_1x^{n-1} + ... + b_n$. Розглянемо систему елементарних симетричних поліномів разом з поліномом $L(X,C)$

$$\begin{cases} e_1 \overset{df}{=} x_1 + x_2 + ... + x_n + b_1 \\ e_2 \overset{df}{=} x_1x_2 + ... + x_{n-1}x_n - b_2 \\ .... \\ e_n \overset{df}{=} x_1x_2...x_n - (-1)^n b_n \\ e_{n+1} \overset{df}{=} L(X,C) - y \end{cases} \quad \textbf{(5.29)}$$

Спираючись на чисто лексикографічний порядок, побудуємо скорочений базис Гребнера поліномів системи (лема 5.3). (Фактично ми виключаємо послідовно змінні множини X , $e_1,...,e_n,e_{n+1}$. Спеціальні алгоритми виключення наведені у [19]). За лемою 5.3, існує елемент цього базису, що не залежить від X. Позначимо його через $P(y,C)$. $P(y,C) \in F[y,C]$. Справді, елементарні симетричні поліноми $e_1,...,e_n$ алгебраїчно незалежні. Останній поліном системи e_{n+1}, оскільки він залежить ще й від змінної y, є також алгебраїчно незалежним від інших поліномів системи. Тому, при виключенні n змінних з системи $n+1$ поліномів, у результаті отримаємо один–єдиний поліном $P(y,C)$. Підстановка в $P(y,C)$ замість y полінома $L(X,C)$ перетворює рівність $P(y,C)=0$ у тотожність:

$$P(L(X,C),C) \equiv 0 \quad \textbf{(5.30)}$$

Тому всі частинні похідні по змінним з C лівої частини рівності (**5.30**) також тотожно дорівнюють нулю.

$$\frac{\partial P(L(X,C),C)}{\partial c_i} \equiv 0, i = 1,..., n.$$

Диференціюючи (**5.30**) як складну функцію, одержимо: $\frac{\partial P}{\partial y} \cdot \frac{\partial L}{dc_i} + \frac{\partial P}{\partial c_i} = 0$

.

Оскільки $\frac{\partial L}{\partial c_i} = x_i$, отримаємо $\frac{\partial P}{\partial y} \cdot x_i + \frac{\partial P}{\partial c_i} = 0$, звідки

$$x_i = -\frac{\partial P}{\partial c_i} \Big/ \frac{\partial P}{\partial y},\ i = 1,..., n.$$

Спеціалізуємо $C = C_0$ так, щоб $\frac{\partial P}{\partial y} \neq 0$. В результаті отримаємо рівності $x_i = -\frac{\partial P}{\partial c_i}(C_0) \Big/ \frac{\partial P}{\partial y}(C_0),\ i = 1,...,n.$, де $\frac{\partial P}{\partial c_i}(C_0), \frac{\partial P}{\partial y}(C_0)$ – поліноми змінної y. Легко бачити, що $\beta = c_1^{(0)} \cdot \alpha_1 + ... + c_n^{(0)} \cdot \alpha_n, c_i^{(0)} \in K, i = 1,...,n$ – шуканий примітивний елемент.

Оскільки поле $F(\beta)$ – просте алгебраїчне розширення F, будь-який ненульовий елемент $F(\beta)$ є оберненим, причому його можна представити у вигляді полінома з $F[\beta]$. Таким чином, $\frac{1}{\partial P/\partial y}(\beta) = Q(\beta)$. Для β виду (**5.28**) через $p_C(y)$ позначимо мінімальний поліном розширення $F(\beta)$ над F. Тому праві частини рівності (**5.29**) можна замінити поліномами $x_i = R_i(\beta),\ i = 1,...,n.$, причому $R_i(\beta)$ можна редукувати за модулем $P_{C_0}(y)$. Тому можна вважати, що

$$\alpha_i = R_i(\beta),\ \deg(R_i) < \deg_y P,\ \ i = 1,...,n. \qquad (\mathbf{5.31})$$

Примітка. Найбільш складним за часом підалгоритмом є алгоритм побудови елемента базису Гребнера $P(y, C)$. Диференціювання у формулі (5.29) виконується за час, поліноміальний відносно n. Спеціалізація $C = C_0$ може бути виконана методом випадкового вибору значень. З ймовірністю 1 буде отриманий вірний результат. Найбільшим «недоліком» цього алгоритму є те, що поліном $P(y, C_0)$, взагалі кажучи, не співпадає з мінімальним поліномом $P_{C_0}(y)$. Оскільки алгоритм обернення елемента $\frac{1}{\partial P/\partial y}(\beta) = Q(\beta)$ у полі $F(\beta)$ використовує $P_{C_0}(y)$, представлення α_i у вигляді (5.10) потребує побудови $P_{C_0}(y)$.

Алгоритм побудови групи Галуа. Нагадаємо, що групою Галуа $G(F, f) \overset{df}{=} G_K(f)$ поля $F_f \overset{df}{=} F(\alpha_1,...\alpha_n)$ розкладання полінома $f(x)$ називається група автоморфізмів поля F_f, що залишає нерухомим основне поле F. Довільний автоморфізм $s \in G_F(f)$ здійснює деяку підстановку коренів полінома f: $S(\alpha_j) = \alpha_{S(j)}$. Таким чином, автоморфізм S можна ототожнити з підстановкою чисел множини $1...n$.

$$s = \begin{pmatrix} 1 & 2 & ... & n \\ j_1 & j_2 & ... & j_n \end{pmatrix} \qquad (\mathbf{5.32})$$

Поняття групи Галуа є центральним поняттям теорії Галуа. Будова цієї групи відіграє основну роль у теорії Галуа, зокрема, для задачі розв'язання

алгебраїчного рівняння в радикалах. Отже, алгоритм побудови групи Галуа має самостійну цінність.

Група Галуа є підгрупою групи усіх підстановок виду (**5.32**), яку називають симетричною групою S_n. Порядок групи S_n дорівнює $n!$. Розглянемо поліном виду $g(y,C,X)=\prod_{k=1}^{n!}(y-s_k(C)\cdot X)$, $\deg_y(g)=n!$. Спеціалізуємо поліном g значенням $C=C_0$ таким, що елемент $C_0\cdot X=\sum_{j=1}^{n}c_j^{(0)}\cdot x_j$ є примітивним. $g_0(y,X)=g(y,C_0,X)$. Поліном $g_0(y,X)$ представимо у вигляді

$$g_0(y,X)=g^{(0)}(y,X)\cdot g^{(1)}(y,X)\cdot\ldots\cdot g^{(m-1)}(y,X) \qquad \textbf{(5.33)}$$

де $m=\frac{n!}{|G_F(f)|}$. (Насправді число m – порядок фактор–групи $S_n/G_F(f)$, тобто кількість класів суміжності S_n за $G_F(f)$). Нехай $G_F^j(f)$ – j–тий клас суміжності. В цих позначеннях $G_F^0(f)=G_F(f)$. За означенням $g^{(j)}(y,X)=\prod_{S_i\in G_F^j(f)}(y-s_i(C_0)\cdot X)$. Спеціалізуємо представлення (**5.33**): $X=A$. Позначимо $g^{(j)}(y,A)=p_j(y)$ Отримаємо $g_0(y,A)=p_0(y)\cdot p_1(y)\cdot\ldots\cdot p_{m-1}(y)$. Доведемо, що $g_0(y,A)=p(y,C_0)$. Усі лінійні комбінації $s_k(C_0)\cdot A$ є коренями полінома $p(y,C_0)$, оскільки будь–яка перестановка коренів S_k переводить систему рівностей (**5.33** 5.6) у себе. Дійсно, поліном $p(y,C)$ можна представити у вигляді

$$p(y,X,C)=h_0\cdot(L(X,C)-y)+h_1\cdot(\sigma_1(X)+b_1)+\ldots+h_n(\sigma_n-(-1)^n b_n) \qquad \textbf{(5.34)}$$

оскільки за означенням базису Гребнера будь–який елемент цього базису можна виразити у вигляді лінійної комбінації елементів висхідної системи поліномів. Спеціалізації $X=A,\ C=s_k(C_0)$ рівності (5.34) усі доданки рівності (5.34), починаючи з другого, перетворюють у 0. Отже, ми отримаємо

$$p(y,A,C_0)=h_0(y,A,s_k(C_o)\cdot(L(A,s_k(C_0))-y)$$

Нехай тепер β – будь–який корінь $p(y,C_0)$. Тоді β є y – координатою розв'язку системи рівнянь (**5.29**). Отже,

$$\beta=c_1^{(0)}\cdot\alpha_1+\ldots+c_n^{(0)}\cdot\alpha_n, c_i^{(0)}\in K,\ i=1,\ldots,n,$$

де $(\alpha_1,\ldots,\alpha_n)$ – вектор коренів $f(x)$. Твердження $g_0(y,A)=p(y,C_0)=P_{C_0}(y)$ доведено. Тому $P_{C_0}(y)$ є дільником $p(y,C_0)$. Таким чином, поліном $p(y,C_0)$,

який отримано алгоритмом побудови скороченого базису Гребнера, має представлення

$$p(y) = p_0(y) \cdot p_1(y) \cdot \ldots \cdot p_{m-1}(y)$$

де усі фактори $p_j(y)$ мають однакові ступені. Один з цих факторів, скажімо, $p_0(y)$ є мінімальним поліномом примітивного елемента $\beta = c_1^{(0)} \cdot \alpha_1 + \ldots + c_n^{(0)} \cdot \alpha_n$ розширення $F(\alpha_1,\ldots,\alpha_n)$ над F. Ступінь цього поліному є порядком групи Галуа. Поліном $p_0(y)$ можна представити у вигляді $p_0(y)) = \prod_{S_i \in G_K(f)} (y - s_i(C_0) \cdot X)$.

Можна вважати, що спеціалізацію $C = C_0$ обрано так, що c_j^0 є попарно різними. Тоді усі підстановки $s_j \in G_K(f)$ можна обчислити. Опишемо алгоритм побудови примітивного елементу розширення $F(\alpha_1,\ldots,\alpha_n)$ над F та відповідної групи Галуа.

Алгоритм побудови примітивного елементу та групи Галуа

Вхід: $f(x) = b_0 x^n + b_1 x^{n-1} + \ldots + b_n$

Вихід:

$q(y)$ – мінімальний поліном розширення $F(\alpha_1,\ldots,\alpha_n)$ над F;

$(c_1^{(0)},\ldots,c_n^{(0)})$ – вектор коефіцієнтів представлення $\beta = c_1^{(0)}\alpha_1 + \ldots + c_n^{(0)}\alpha_n$;

$h_1(y),\ldots,h_n(y)$ – поліноми представлення коренів α_j через β: $\alpha_j = h_j(\beta)$;

$s_1,\ldots,s_m$ – множина підстановок – елементів групи Галуа.

Початок

1: Обрати випадковим чином n попарно різних натуральних чисел $(c_1^{(0)},\ldots,c_n^{(0)})$;

Скласти набір поліномів

$$\begin{cases} e_1 \overset{df}{=} x_1 + x_2 + \ldots + x_n + b_1 \\ e_2 \overset{df}{=} x_1x_2 + \ldots + x_{n-1}x_n - b_2 \\ \ldots\ldots \\ e_n \overset{df}{=} x_1x_2\ldots x_n - (-1)^n b_n \\ e_{n+1} \overset{df}{=} c_1^{(0)}x_1 + \ldots + c_n^{(0)} - y \end{cases}.$$

Побудувати скорочений базис Гребнера $(g_1,...,g_k)$ цього набору, використовуючи лексикографічне упорядкування змінних $x_1 \succ ...x_n \succ y$;

$p(y) := g_k(y)$; // $g_k \in K[y]$;

$Deg := \deg(g_i)$;

Якщо $Deg < n!$ перейти до 1, інакше

Розкласти поліном $p(y)$ на незвідні множники

$$p(y) = p_0(y) \cdot ... \cdot p_{m-1}(y).$$

Підалгоритм пошуку мінімального поліному $q(y)$*:*

Знайти поліном $p_j(y)$, такий, що $p_j(y - \sum c_j^{(0)} \alpha_j) = 0$;

$q := p_j$;

Знайти скорочений базис Гребнера $(l_1,...,l_n)$ системи

$$(g_1,...,g_n,q); \qquad \textbf{(5.35)}$$

//Елементи цього базису мають вигляд $l_j = x_j - h_j(y), \deg(h_j) < \deg(p)$

$k := 1$;

Для усіх $s \in S_n$

Якщо $q(y - s(C_0) \cdot X) \in Ideal(l_1,...,l_n,q)$

$s_k := s,\ k := k+1$

Кінець

Підалгоритм пошуку мінімального полінома $q(y)$*:*

Знаходить поліном $p_j(y)$, що виконується умова $p_j(y - \sum c_j^{(0)} \alpha_j) = 0$;

Вхід:

$(c_1^{(0)},...,c_n^{(0)})$ – вектор коефіцієнтів представлення $\beta = c_1^{(0)}\alpha_1 + ... + c_n^{(0)}\alpha_n$;

$p_0(y), p_1(y),..., p_{m-1}(y)$ – фактори розкладу $p(y) = p_0(y) \cdot p_1(y) \cdot ... \cdot p_{m-1}(y)$.

*Вихід***:**

$q(y)$ – мінімальний поліном розширення $K(\alpha_1,...,\alpha_n)$ над K;

Початок

1: Оберемо достатньо велике просте число M. Через F_M позначимо скінченне просте поле з M елементів.

Розглядаючи $f(x)$ як елемент кільця $F_M[x]$, знайдемо усі корені $\mu_1,...,\mu_n$ $f(x)$ у F_M.

Обчислимо число $d = c_1^{(0)}\mu_1 + ... + c_n^{(0)}\mu_m$ як елемент поля F_M.

Обчислимо $p_0(d),..., p_{m-1}(d)$ та знайдемо такий індекс j, для якого $p_j(d) = 0$. Легко бачити, що таке число завжди існує. Якщо число j єдине, $p_j(y)$ є шуканим. Інакше треба змінити модуль обчислень M та повторити обчислення, тобто перейти до п.1.

Кінець

Суттєвим моментом для обґрунтування цього алгоритму, який ми ще не розглянули, є побудова поліномів $h_j(y)$ таких, що $\alpha_j = h_j(\beta)$. Зауважимо, що набір поліномів

$$x_1 - h_1(y), \ldots, x_n - h_n(y), p(y) \qquad (\mathbf{5.36})$$

за означенням, є скороченим базисом Гребнера. Тому, в силу унікальності скороченого базису Гребнера (при фіксованому упорядкуванні) при виконанні кроку (5.35) буде дійсно побудовано набір виду (5.36).

Примітка. Алгоритм, який ми щойно розглянули, є дуже складним за часом. Основною причиною є той факт, що ступінь полінома $p(y, C_0)$, а, як наслідок, і ступінь $q(y)$, і порядок групи Галуа, оцінюється величиною $n!$. Таким чином, алгоритм побудови базису Гребнера має справу з поліномами дуже великих ступенів. Це робить алгоритм придатним лише для досить малих значень n. Отже, він має здебільшого теоретичне значення.

5.4. Методи відділення та уточнення дійсних коренів полінома

Базиси Гребнера є основним інструментом розв'язання багатьох задач комп'ютерної алгебри, основою яких є розв'язання та дослідження систем алгебраїчних рівнянь у полях комплексних чисел. Задачі розв'язання та дослідження алгебраїчних рівнянь у полях дійсних чисел використовують алгоритми відокремлення та уточнення дійсних коренів.

Нехай $f(x)$– поліном з коефіцієнтами, які є елементами конструктивного дійсного поля F. Для простоти ми будемо вважати, що F є полем раціональних чисел Q. Класична задача відокремлення дійсних коренів $f(x)$ полягає у побудові послідовності таких числових проміжків, що, по-перше, попарно не перетинаються, і, по–друге, кожен з яких містить у точності один дійсний корінь $f(x)$. Задача уточнення дійсних коренів $f(x)$ полягає у побудові такої послідовності числових проміжків, довжина кожного члена якої не перевищує заданого додатного раціонального числа ε. Досить повний огляд алгоритмів відокремлення та уточнення дійсних коренів полінома $f(x)$ наведено у [72]. Сформулюємо найбільш відомий з таких алгоритмів – алгоритм Штурма [7, 8].

5.4.1. Алгоритм Штурма

Нехай $f(x) \in Q[x]$ – поліном, вільний від квадратів і $a, b \in Q$, причому $a < b$. Послідовністю Штурма для f на проміжку $[a, b]$ називається послідовність поліномів $f_0, f_1, \ldots, f_k$ така, що

1. $f_0 = f$, $f_1 = f'$;

2. $f_{i+2} = -f_i \bmod f_{i+1}$;

3. $f_k = d, d \in Q, d \neq 0.$

Нехай $c \in [a,b]$. Через $W(c)$ позначимо кількість змін знаків у послідовності $\{f_0(c).f_1(c),..., f_k(c)\}$.

Теорема 5.6 *(Штурма).*

Кількість дійсних коренів полінома $f(x)$ на $[a,b]$ дорівнює $W(a) - W(b)$.

Ідея алгоритмів відділення та уточнення дійних коренів полягає у наступному: припустимо, що множина усіх дійсних коренів $f(x)$ належить $[a,b]$.

1. Обчислимо послідовність Штурма для $f(x)$.

2. Обчислимо $R(a,b) = W(a) - W(b)$.

3. Оберемо точку $c = (a+b)/2$ та обчислимо

$(R(a,c) = W(a) - W(c),\ R(c,b) = R(a,b) - R(a,c)$

Тим самим ми уточнили кількість коренів на половинах $[a,c]$, $[c,b]$ та відрізку $[a,b]$. Повторюючи п.3 для отриманих половинних відрізків, на деякому кроці отримаємо послідовність відрізків, кожен з яких містить один дійсний корінь $f(x)$. Для уточнення коренів, які вже відокремлені, можна використати загальновідомі швидкі алгоритми чисельного аналізу, наприклад, алгоритм хорд та дотичних.

5.4.3. Задача дослідження функції дійсного аргументу

У цьому пункті розглянемо методи та алгоритми розв'язання однієї з найбільш відомих навчальних задач як шкільного курсу алгебри та початків аналізу, так і курсу основ математичного аналізу у ВНЗ – задачу дослідження функції дійсного аргументу методами диференціального числення. У шкільному курсі алгебри та початків аналізу цю задачу розглядають для функцій одного аргументу, у вишах – для функцій кількох (двох, трьох) аргументів.

Зауважимо, що у загальному формулюванні ця задача є досить складною. Тому ми обмежимося її частинним, але практично важливим випадком, у якому функцію $y = F(x_1,...,x_n)$ F задано у радикалах з дійсними коефіцієнтами.

Постановка задачі. Нехай $X = (x_1,...x_n)$ – вектор змінних. Розглянемо функцію дійсного аргументу $y = f(X)$, тобто $x_j, y \in R$, і R – поле дійсних чисел. Задача дослідження функції $y = f(X)$ є комплексною. Вона включає декілька конкретних задач:

1. Знайти область визначення $y = f(X)$.
2. Знайти область значень $y = f(X)$.
3. Знайти нулі $y = f(X)$ та інтервали знакосталості.

4. Знайти критичні точки $y = f(X)$ (точки екстремумів, точки розривів) та інтервали монотонності.

5. Знайти точки перегинів та інтервали опуклості.

Кожна з цих задач формулюється у термінах елементарної теорії дійно–замкнених полів [74, 75] як замкнена формула відповідної прикладної логіки предикатів.

Наприклад, для функції $y = \frac{2 \cdot x + 1}{\sqrt{x^2 - 1} - 4}$ задача п.1 (пошуку області визначення) формулюється як наступна формула елементарної теорії дійно–замкнених полів $\exists x \exists z (z^2 = x^2 - 1) \,\&\, (z \geq 0) \,\&\, (z - 4 \neq 0)$.

Отже, ми маємо справу з навчальною математичною задачею від декількох змінних. Для зведення цієї задачі до задачі однієї змінної треба елімінувати квантор $\exists z$ та застосувати відповідні алгоритми.

Розглянемо наступну задачу.

***Задача* 5.3**.

Нехай $y = F(X)$ – функція n дійсних змінних $x_j \in [m_j, M_j]$. Визначити, чи належить значення змігнної y заданому відрізку $[m_y, M_y]$.

Алгоритми, розглянуті нижче, істотно залежать від виду функції $F(x)$. Якщо $F(x)$ є поліномом, задачу будемо називати визначеною поліноміально. Якщо $F(x)$ – раціональна функція, задачу будемо називати визначеною раціонально. Якщо ж формула F містить радикали, задача є радикально визначеною.

Розглянемо спочатку поліноміально визначені задачі, потім – раціонально визначені задачі, і, нарешті – радикально визначені задачі. Будемо також вважати, що коефіцієнти f_i – суть раціональні числа. Отже, $F \in Q[X]$ для поліноміально визначених задач, $F \in Q(X)$ для раціонально визначених задач.

Кожній змінній x_i припишемо область її визначення – числовий відрізок $D_i = [m_i; M_i]$, $m_i, M_i \in Q$. Через D позначимо декартовий добуток $D_1 \times ... \times D_n$. Для функції $f(x)$ через $S(D)$ позначимо числову множину, визначеною формулою

$$S(D) = \{b : b = f(a), a \in D\}.$$

Отже, $S(D)$– множина значень змінної y, якщо аргументи функції пробігають усю область визначення. Для поліноміально визначених задач $S(D)$ є числовим відрізком.

Математичне обґрунтування алгоритму. Задачу 5.3. ми будемо розв’язувати класичними методами математичного аналізу, використовуючи

метод визначення найбільших і найменших значень неперервної функції $y = f(x_1,...,x_n)$ на замкнутій обмеженій множині.

1. *Метод побудови системи алгебраїчних рівнянь для координат критичних точок відображення.* Позначимо через $a = (a_1,...a_n)$ точку R^n. Нехай у точці $a \in D$ функція $f(X)$ досягає найбільшого (найменшого) значення. Тоді

1. або $\frac{\partial f}{\partial x_j}(a) = 0,\ j = 1,...,n,$

(**5.37**)

2. або точка a належить границі паралелепіпеда D.

Таким чином, алгоритм пошуку критичних екстремальних точок функції $f(X)$ на множині D може бути заснований на переліченні всіх можливих варіантів. Кожний з варіантів визначений підмножиною координат, для яких координатні змінні приймають граничні значення $x_j = m_j$ або $x_j = M_j$. Назвемо такі координати *граничними*. Для інших координат (назвемо їх *внутрішніми*) у цьому варіанті потрібно розглядати систему співвідношень (**5.37**). Загальна кількість варіантів дорівнює 3^n.

Будемо вважати, що у даному варіанті необхідні граничні значення змінних уже підставлені в поліном f і результат приведений до канонічного виду. Цей результат ми також будемо позначати через f. Покладемо $\frac{\partial f}{\partial x_j} = g_j(X)$, $X = (x_{i1},...,x_{ik})$. (На практиці корисно звільнити поліноми $g_i(X)$ від квадратів.) Отже, (**5.37**) є системою цілих алгебраїчних рівнянь $g_j(X) = 0,\ j = 1,...,k$.

2. *Метод побудови мінімального полінома для k–ої координати множини критичних точок.* Побудуємо базис Гребнера ідеалу $(g_1,...,g_n)$ для чисто лексикографічного порядку, елімінуючи послідовно змінні $x_{i1},...,x_{ik-1}$. Для простоти позначень покладемо $u_j = x_{i_j}$. За лемою 5.3, базис Гребнера має «трикутний» вид. Запишемо відповідну систему рівнянь:

$$(g_1(u_1,u_2,...,u_k) = 0)\ \&\ ...\ \&\ (g_k(u_k) = 0) \qquad (\mathbf{5.38})$$

Система (5.38) рівносильна системі (5.37), а дійсні корені $r_1,...,r_l$ рівняння $g_k(u_k) = 0$ – суть відповідні координати критичних екстремальних точок f.

3. *Метод відділення координат критичних точок на відрізку*. Ці корені можуть бути відділені на відрізку $[m_k;\ M_k]$ з наперед заданою точністю за допомогою, наприклад, алгоритму Штурма (рис. 5.1)

На ньому позначені корені r_j й кінці відрізків $[a_j; b_j]$, що відокремлюють ці корені. Для кожного $\delta > 0$ можна отримати такі відрізки, що $|b_i - a_i| < \delta$.

Оскільки індекс k може бути обраний довільно, аналогічну процедуру можна застосувати до усіх внутрішніх координат.

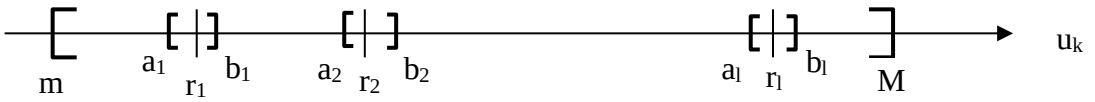


Рис. 5.1. Відділення коренів $g_k(u_k)$.

4. Метод побудови образів множин критичних точок і кандидатів у критичні точки відображення. Розглянемо множину V точок виду $(r_{1j_1},...,r_{kj_k})$ простору R^k, кожна координата яких – дійсний корінь відповідного полінома $g_i(u_i) = 0,\ i = 1,...k$ на відрізках $D_i = [m_i; M_i]$. Ці точки – суть кандидати у критичні точки полінома $f(X)$ на проекції паралелепіпеда D на підпростір, визначений внутрішніми координатами. Пояснимо ситуацію ілюстрацією на графіку для $k = 2$:

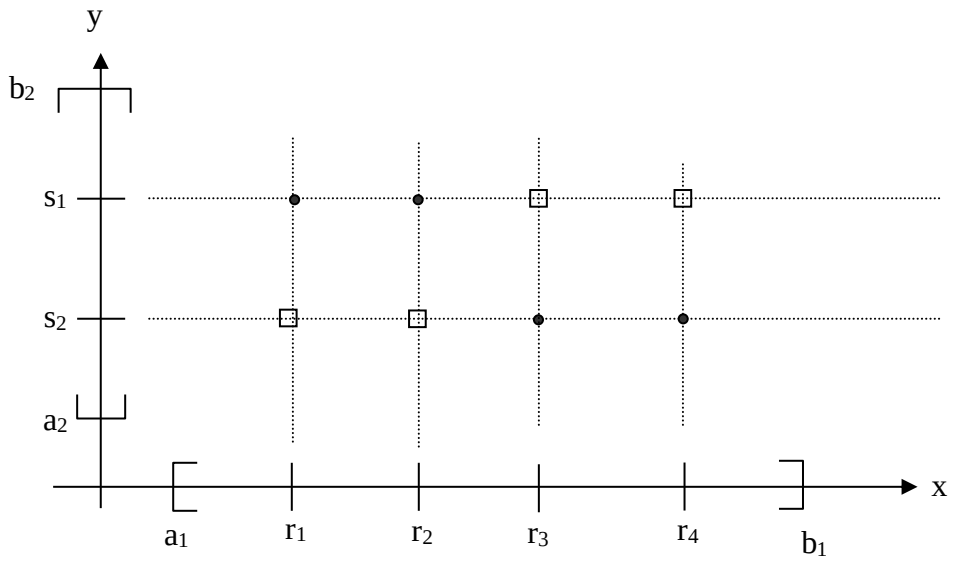


Рис. 5.2. Множина точок – кандидатів у критичні точки.

Припустимо, що отримана трикутна система рівнянь має вигляд $g_1(x, y) = 0,\ g_2(y) = 0$ причому $\deg_x g_1 = \deg_y g_1 = \deg_y g_2 = 2$ й усі її корені дійсні. Тоді множину розв'язків системи можна записати у вигляді $(s_1, r_1), (s_1, r_2), (s_2, r_3), (s_2, r_4)$ (на рис. 5.2 – чорні точки). В результаті перетворення системи до діагонального виду одержимо систему $g_1^{'}(x) = 0,\ g_2(y) = 0$, причому $\deg_x g_1^{'} = 4,\ \deg_y g_2 = 2$. Тому кандидатами в критичні точки будуть усі пари $(s_i, r_j),\ i = 1,2;\ j = 1,2,3,4$. На рис. 5.2 бачимо, що

серед кандидатів у критичні точки 4 точки є помилковими – вони відзначені квадратиками.

Проте, оскільки всі кандидати в критичні точки належать паралелепіпеду D, задача $S(D) \subset D_i$ еквівалентна задачам $\forall a \in V \;\; f(a) \in D_i$.

Розглянемо систему рівнянь

$$\begin{aligned} & y = f(u_1,...,u_k) \\ & g_1(u_1,u_2,...,u_k) = 0 \\ & g_2(u_2,...,u_k) = 0 \\ & ... \\ & g_k(u_k) = 0 \end{aligned} \qquad (\mathbf{5.39})$$

Елімінуючи послідовно в цій системі змінні $u_1,...,u_k$ методом побудови базису Гребнера, отримуємо поліном – елемент базису $h(y)$. Легко бачити, що серед коренів цього поліному – образи множини критичних точок $\{f(a): a \in V\} \subseteq \{y: h(y) = 0\}$.

Розглянемо знову систему рівнянь (**5.38**). Оскільки ця система зведена до трикутного виду, до неї можна застосувати процедуру діагоналізації, виключаючи недіагональні змінні тим же методом базисів Гребнера. У результаті одержимо систему

$$\begin{aligned} & y = f(u_1,...,u_k) \\ & g_1'(u_1) = 0 \\ & g_2'(u_2) = 0 \\ & ... \\ & g_k(u_k) = 0 \end{aligned} \qquad (\mathbf{5.40})$$

Елімінуючи послідовно змінні $u_1,...,u_k$ з системи (**5.40**) методом побудови базису Гребнера (як у попередньому випадку), отримуємо поліном базису $h_1(y)$. Легко побачити, що серед коренів цього поліному – всі кандидати в критичні точки.

5. *Метод побудови відповідності «множина критичних точок – образ цієї множини».* Проблема полягає в тому, щоб співвіднести до кожного елементу множини корінь $h(y)$ числа виду $f(a)$. Розглянемо рис. 5.3, що ілюструє сказане вище.

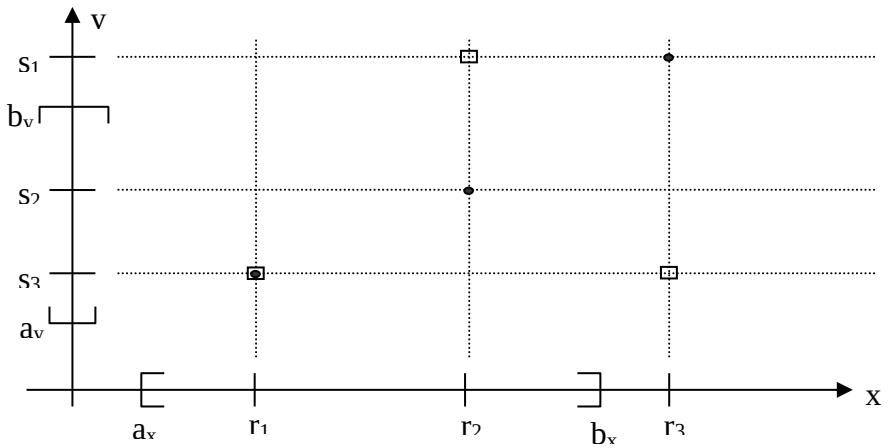


Рис. 5.3. Множина критичних точок та її образ.

Апріорі можливі кілька варіантів реалізації співвідношення $s_{i_j} = f(r_j),\ j = 1,2.3.$ Один із цих варіантів показаний на рис. 5.3 чорними кружками, другий – незаштрихованими квадратиками.

1–ий варіант: $s_1 = f(r_1),\ s_2 = f(r_2),\ s_3 = f(r_3).$

2–ой варіант: $s_1 = f(r_1),\ s_2 = f(r_3),\ s_3 = f(r_2).$

У першому варіанті з $r_i \in [a,\ b] \Rightarrow s_i \in [a, b]$. У другому варіанті ця властивість не виконується.

Нехай $y = f(X) \in Q[X]$ і $X', X'' \in R^n, y' = f(X'),\ y'' = f(X'')$. Тоді

$$y'' - y' = \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}(\bar{X}_i)(x_i'' - x_i')$$

де $\bar{X}_i$ – деякий набір точок з паралелепіпеда з головною діагоналлю X', X''.

Звідси випливає нерівність $\left|y'' - y'\right| \le \sum_{i=1}^{n} \left|\frac{\partial f}{\partial x_i}(\bar{X}_i)\right| \left\lfloor x_i'' - x_i' \right\rfloor$. Значення частинних похідних можна ефективно оцінити через значення границь паралелепіпеда D: $\left|\frac{\partial f}{\partial x_i}(\bar{X}_i)\right| \le C_i(D)$. Тому

$$\left|y'' - y'\right| \le \sum_{i=1}^{n} C_i \left\lfloor x_i'' - x_i' \right\rfloor \qquad \textbf{(5.41)}$$

Нерівність (5.41) будемо використовувати для визначення відповідності $s_{i_j} = f(r_j),\ j = 1,...,l)$ у такий спосіб: нехай $U = \{s_1,..., s_l\}$ – множина розв'язків системи (5.13) і $V = \{A,..., A_s\}$ відповідно точки множини V.

6. *Основний алгоритм*

1. Відокремимо точки множини V паралелепіпедами E_j такими, що $A_i \in E_j$, а довжини ребер не перевершують даного додатного числа δ.

2. Відокремимо також точки s_j множини U відрізками $[a_j; b_j]$, довжини яких не перевершують даного додатного числа ε.

3. Виберемо в паралелепіпеді E_j точку $r_j^{'}$ з раціональними координатами й обчислимо $s_j^{'} = f(r_j^{'})$. Використаємо нерівність (5.15) для точок $r_j, r_j^{'}$ у правій частині (5.27): $\left|s_{i_j} - s_j^{'}\right| \le \sum_{i=1}^{n} C_i \left\lfloor x_i^{''} - x_i^{'} \right\rfloor \le \delta \sum_{i=1}^{n} C_i$. Таким чином, якщо вибрати $\varepsilon \le \delta \sum_{i=1}^{n} C_i$, точка $s_j^{'} = f(r_j^{'})$ буде стояти від відділеного кореня s_{i_j} на відстань, меншу за ε. Для заключного висновку найкраще проілюструвати описану ситуацію на числовій осі:

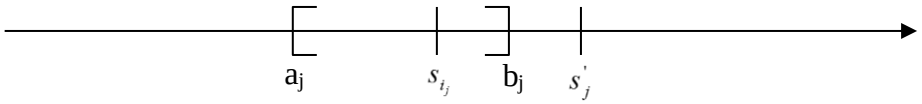


Рис. 5.4. Правило 3–х епсілон.

Таким чином, точка $s_j^{'} = f(r_j^{'})$ має належати відрізку $[a_j - \varepsilon;\ b_j + \varepsilon]$.

Правило 3–х епсілон. Якщо зафіксувати таке мале додатне число ε, що при відділенні кореня полінома $h_1(y)$ відрізки $[a_j - \varepsilon;\ b_j + \varepsilon]$ не будуть перетинатися й не будуть містити кінців відрізка D_y, корені поліномів $g_k(u_k)$ потрібно відокремити з точністю $\delta = \varepsilon / \sum C_j$. Тоді знаходження образу коренів здійснюється за допомогою основного алгоритму, у якому ε, δ обрані за правилом 3–х епсілон.

4. Якщо корінь полінома $h_1(y)$ й точки множини кандидатів у критичні точки V відділені з дотриманням правила 3–х епсілон, всі точки $s_j^{'} = f(r_j^{'})$ мають належати відрізку $D_i = [m_i; M_i]$. Тоді y належить заданому відрізку. Інакше виконується заперечення цього співвідношення.

Раціонально визначені задачі. У цьому випадку формула функції має вид $y = F(X)/H(X)$, $X = \{x_1, ..., x_n\}$, $F, H \in K[X]$. Якщо границі y $[m_y, M_y]$ скінченні, поліном $H(X)$ на області D не повинен мати дійсних коренів. Таким чином, перший етап алгоритму має відокремити усі корені полінома $H(X)$ таким чином, щоб границі області D не входили до відрізків, що відокремлюють корені.

Якщо відокремлюючий паралелепіпед деякого дійсного кореня знаменника $H(X)$ входить до D, функція $y = F(X)/H(X)$ необмежена на D. В іншому разі (другий етап) задача розв'язується цілком аналогічно до

поліноміально визначеної задачі, причому у якості поліномів $g_i(X)$ треба брати поліноми

$$\frac{\partial F}{\partial x_i}\cdot H(X)-\frac{\partial H}{\partial x_i}\cdot H(X)$$

Алгебраїчно визначені задачі. Для радикально визначених задач формула функції $y=F(X)$ містить радикали: $F(X)=G(X,\sqrt[k]{H(X)})$. Цей вираз можна перетворити наступним чином: $G(X,\sqrt[k]{H(X)})\sim G(X,u)\,\&\,(u^k=H(X))$.

В наших задачах для парних значень додатково треба врахувати невід'ємність підкореневого виразу. Таким чином, для задач над дійсними числами маємо $G(X,\sqrt[2k]{H(X)})\sim G(X,u)\,\&\,(u^k=H(X))\,\&\,H(X)\ge 0$. Отже, функцію $F(X)$ на області D або на області $D\,\&\,H(X)\ge 0$ визначено таким чином:

$$(y=G(X,u))\,\&\,(u^k=H(X)) \qquad (\mathbf{5.42})$$

Можна дещо узагальнити (5.42) вважаючи, що другий кон'юнкт має загальний вид поліноміальної рівності від змінних X,u: $(y=G(X,u))\,\&\,(P(u,X)=0)$. Тоді частинні похідні $y=G(X,u)$ на $P(X,u)=0$ обчислюються з формул

$$(\partial F/\partial x_j=\partial G/\partial x_j+\partial G/\partial u\cdot\partial u/\partial x_j)\,\&\,(\partial P/\partial x_j+\partial P/\partial u\cdot\partial u/\partial x_j=0)$$

Отже,

$$(\partial G/\partial x_j+\partial G/\partial u\cdot\partial u/\partial x_j=0)\,\&\,(\partial P/\partial x_j+\partial P/\partial u\cdot\partial u/\partial x_j=0) \qquad (\mathbf{5.43})$$

З системи рівнянь (5.43) можна виключити змінну u, побудувавши скорочений базис Гребнера. В результаті отримаємо рівняння $g_j(X)=0$, яке грає роль рівняння (**5.37**) в алгоритмі розв'язання нашої задачі. Зауважимо, що наш основний алгоритм є досить неефективним.

Застосування алгоритму у програмах фізичних обчислень. Задача про границі зміни фізичних величин. Результати обчислень, здійснюваних програмою, використовуються в реальних фізичних приладах і пристроях. Тому значення змінних не можуть бути довільними – вони мають перебувати в границях, установлених параметрами приладу або пристрою. Тому помилки в програмах, пов'язані з виходом значень змінних за межі припустимих значень, також не можуть бути виявлені ні під час компіляції, ні під час виконання програми. Цю задачу також потрібно вирішувати методами статичного аналізу програмного коду. Користувач має специфікувати всі

змінні програми границями їхніх припустимих значень, а аналізатор повинен перевірити правильність цієї специфікації.

5.5. Системи лінійних нерівностей

У даному пункті розглянуто задачу розв'язання системи лінійних нерівностей. Алгоритм розв'язання системи лінійних нерівностей оснований на підході глави 3. Результатом застосування цього підходу є алгебраїчні специфікації відповідних предметних областей і алгоритмів через спадкування, розширення й морфізмів багатосортних алгебраїчних систем.

Підкреслимо, що специфікаціям підлягають власно предметні області, а не тільки алгоритми. Зокрема, алгоритм розв'язання системи лінійних нерівностей, наведений нижче, будує канонічну форму системи лінійних нерівностей. Крім цього, розглядаються і інші алгоритми: алгоритм зведення кратного інтеграла на багатогранній області до повторних інтегралів, алгоритм зміни порядку інтегрування, а також алгоритм розв'язання задачі лінійного програмування (пошук максимуму лінійної функції на багатогранній області).

Канонічна форма використовує ідею проектування (елімінації квантора) методів Фур'є–Моцкина [77], Фур'є–Чернікова [78].

5.5.1. Системи лінійних нерівностей (СЛН). Загальні відомості

Лінійна нерівність (ЛН) L має вигляд

$$L = a_1 x_1 + a_2 x_2 + ... + a_m x_m \le b, \quad x_j \in Variable, \quad a_j, b \in Coef$$

$$A = (a_1, ..., a_m), X = (x_1, ..., x_m). \; L = A \cdot X \le b.$$

Таким чином, алгебра лінійних нерівностей використовує лінійний простір *LinComb* над упорядкованим полем *Coef*, а також алгебру змінних *Variable*.

Системи лінійних нерівностей (СЛН) задаються логічними формулами виду $S = L_1 \& L_2 ... \& L_n$, де L_j лінійні нерівності.

Основні задачі теорії СЛН:

1. Перевірити виконуваність (існування розв'язку).
2. Привести до канонічного виду (розв'язати).

Зауважимо, що розв'язком є полігон (опуклий багатогранник) в $n-$вимірному просторі. Отже, канонічна форма має описувати багатогранник розв'язків – наприклад, через вершини (вектори або точки 0–граней) або рівняння $n-1$ граней. Наш спосіб канонічної форми викладено нижче.

Особлива нормальна форма ЛН і СЛН – x-розв'язувана форма.

Нехай x – змінна, що входить до лівої частини ЛН із ненульовим коефіцієнтом. Позначимо $Y = X \setminus \{x\}$. Тоді ЛН можна перетворити до виду

$$x \le B \cdot Y + b \text{ або } x \ge B \cdot Y + b.$$

Таку форму ЛН будемо називати розв'язуваною відносно x або x-розв'язуваною. Якщо кожну лінійну нерівність системи, що залежить від x, представлено в розв'язуваній формі, таку СЛН ми будемо називати представленою в x-розв'язуваній формі. Будь–яку СЛН можна представити в x–розв'язуваній формі. Більш точно, СЛН можна представити у вигляді $LL(x) \& GL(x) \& FL$, де кожна нерівність $LL(x)$ має вигляд $x \le B \cdot Y + b$, кожна нерівність $GL(x)$ має вигляд $x \ge B \cdot Y + b$, кожна нерівність FL не залежить від x.

Відомий алгоритм Фур'є-Моцкіна перевірки сумісності СЛН [77] і його вдосконалення – метод Фур'є-Чернікова [78], заснований на наступному факті:

СЛН $(x \le B_1 Y + b_1) \& (x \ge B_2 Y + b_2)$ сумісна тоді й тільки тоді, коли сумісна ЛН $B_2 Y + b_2 \le B_1 Y + b_1$. У теоретико–логічному формулюванні:

$$\exists Y \exists x (x \le B_1 \cdot Y + b_1) \& (x \ge B_2 \cdot Y + b) \sim \exists X (B_1 \cdot Y + b_1 \ge B_2 \cdot Y + b_2)$$

це перетворення по суті елімінує квантор $\exists x$. Геометрично це перетворення полягає в проектуванні області розв'язків на підпростір $W(Y)$.

Нехай дані дві лінійні нерівності одного знаку: $L_1 = x \le B_1 \cdot Y + b_1,\ L_2 = x \le B_2 \cdot Y + b_2$. Тоді множина розв'язків $L_1 \& L_2$ описується формулою

$$L_1 \& L_2 = \{x : x \le Min(B_1 \cdot Y + b_1, B_2 \cdot Y + b_2)\}$$

Аналогічно, для лінійних нерівностей протилежних знаків отримуємо

$$G_1 \& G_2 = \{x : x \ge Max(B_1 \cdot Y + b_1, B_2 \cdot Y + b_2)\}$$

Через $R(x, S, L, G)$ позначимо множину розв'язків системи ЛН протилежних знаків $L \& G$ на числовій множині S осі Ox. Тоді

$$R(x, S, L_1, G_1) \cap R(x, S, L_2, G_2) = R(x, S, \min(L_1, L_2), \max(G_1, G_2))$$

На рис. **5.5** показано, що числовий проміжок проекціями точок перетину $A_1 = L_1 \cap L_2, A_2 = R_1 \cap R_2$ розбивається на три таких частини, що на кожній із них множина розв'язків описується формулою $R(x, S, L, G)$.

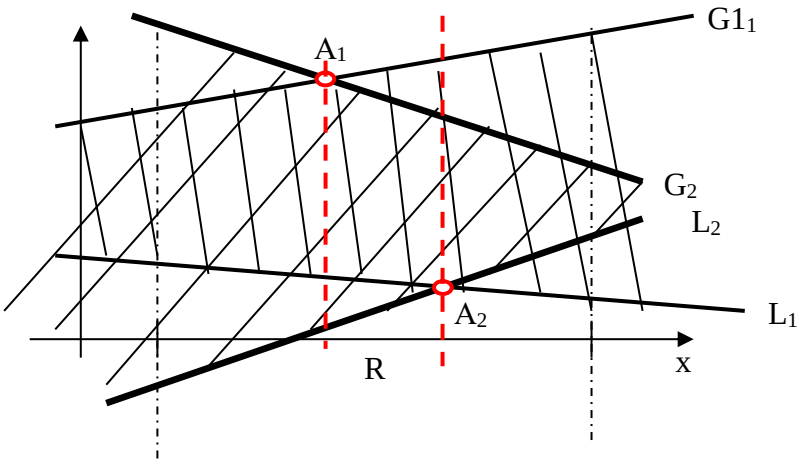


Рис. 5.5. Область розв’язків СЛН типу $L \& G$.

5.5.2. Окремий випадок: розв’язання СЛН на площині

Задача побудови області розв’язків СЛН широко відома [79]-[82]. У випадку, коли $n = 2$, ця задача може бути сформульована як задача побудови опуклої лінійної оболонки за СЛН, що задає сторони багатокутника розв’язків – однієї з основних задач обчислювальної геометрії й комп'ютерної графіки [83], [84]. В лінійному програмуванні ця задача може бути сформульована як реалізації графічного методу лінійного програмування [79].

Основна ідея ілюструється рис. 5.5–5.7. Припустимо, що багатокутник розв’язків R, кожна сторона якого задана ЛН, розбитий на області – трапеції R_1, R_2, R_3, R_4. $R = R_1 + + R_2 + + R_3 + + R_4$. (див. рис. 5.6). Додавання нової ЛН до СЛН полягає у тому, що R_2 й R_4 розбиваються на дві частини, а в R_3 змінюється верхня сторона (шапка): $R_2 = R_2^{'} + + R_2^{''}$, $R_4 = R_4^{'} + + R_4^{''}$, $R_3 = R_3^{1}$. В результаті перетворення отримаємо: $R^{'} = R_1 + + R_2^{'} + + R_2^{''} + + R_3^{'} + + R_4^{'} + + R_4^{''}$.

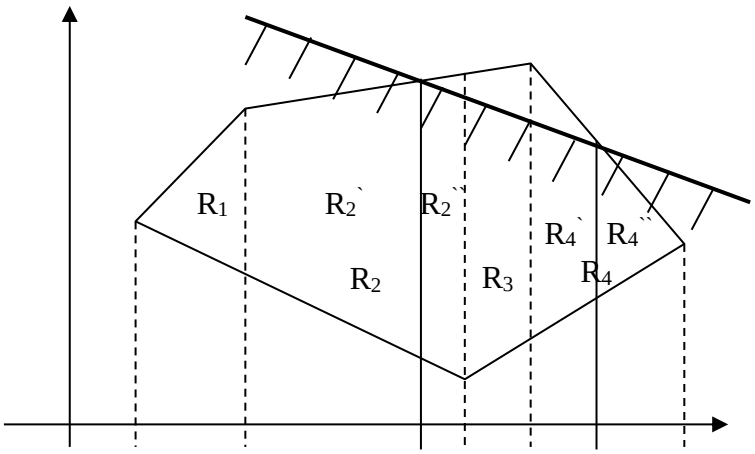


Рис. 5.6. Метод розв’язання СЛН: додавання нової нерівності.

Додавання нової нерівності може призвести до злиття декількох областей в одну (рис 5.7).

Таким чином, алгоритм послідовно додає до вже побудованого багатокутника новий півпростір, перетинаючи його з уже побудованим багатокутником.

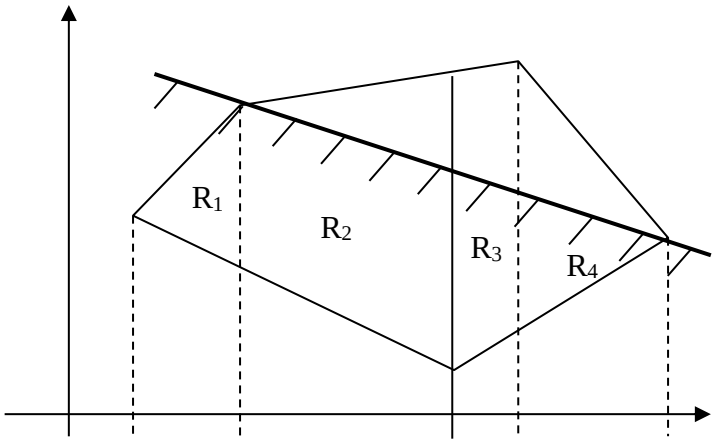


Рис. 5.7. Додавання нової нерівності до СЛН.

5.5.3. Конструктивна алгебра СЛН - алгебра опуклих багатогранників

Суть нашого підходу до задачі розв'язання СЛН полягає в наступному. По-перше, ми визначимо таку вихідну багатосортну алгебраїчну систему СЛН A_{Init}, в якій СЛН є виразом у сигнатурі A_{Init}. По-друге, ми визначимо спеціальну конструктивну багатосортну алгебраїчну систему СЛН A_{Constr}, тобто визначимо *конструктори елементів* і *інтерпретатори операцій* у вигляді систем правил переписувань. По-третє, ми визначимо ізоморфізми

$$\mu : A_{Init} \to A_{Constr}\ ,\ \mu^{-1} : A_{Constr} \to A_{Init}\ .$$

Обчислення значення $Val(F(X))$ виразу $F(X)$ будуються за схемою $Val(F(X)) = \mu^{-1}(Can(F(\mu(X)))$.

Наведемо змістовні описи алгебр A_{Init} і A_{Constr} .

1. Багатосортна алгебра СЛН A_{Init} використовує наступні сорти:
   - Сорт *Coef* – лінійно впорядковане числове поле.
   - Сорт *Variable* – впорядкована множина змінних.
   - Сорт *LinComb* – векторний простір афінних лінійних комбінацій.
   - Сорт *LinUnEqu* – алгебра лінійних нерівностей.
   - Сорт *SysLinUnEqu* – алгебра систем лінійних нерівностей.
2. Для опису конструктивної багатосортної алгебри СЛН A_{Const} треба визначити такі сорти:

Сорти Coef і *Variable* як базові сорти. Ці сорти, як правило, реалізовані в спеціалізованих системах програмування.

Сорт LinUnEqu. Елементами цього сорту є лінійні нерівності, представлені в x–розв'язуваній формі, де x – старша за порядком змінна лінійної нерівності.

$$x_n \le a_{n-1} \cdot x_{n-1} + ... + a_1 \cdot x_1 + b_1 \quad \text{або} \quad x_n \ge a_{n-1} \cdot x_{n-1} + ... + a_1 \cdot x_1 + b_1$$

Операція *XCan(L)* перетворює довільний вираз $L = B_1 \cdot X + b_1 \le B_2 \cdot X + b_2$ у сигнатурі алгебри лінійних нерівностей до цієї канонічної форми.

Сорт Interval – алгебра числових проміжків на розширеній числовій осі. Розширення числової осі здійснюється додаванням до лінійно–упорядкованого числового поля *Coef* символів **–*Inf*, +*Inf*** (мінус нескінченність, плюс нескінченність). Розширену числову вісь ми будемо називати *ExtCoef*.

В алгебрі *Interval* числових проміжків на розширеній числовій осі визначені анулюючий елемент – пустий *відрізок* O і нейтральний *елемент* $I = [-Inf, +Inf]$**.** За визначенням, для будь–якого елемента $c \in Coef$ має місце подвійна нерівність $-Inf < c < +Inf$. Відносно операції перетинання алгебра *Interval* утворює ідемпотентну комутативну напівгрупу з нулем та одиницею.

Крім операції перетинання відрізків, позначеною символом "&", на сорті *Interval* визначена операція розбиття відрізка:

$$Partition([a,b],c) = [a,c] + +[b,c]\,.$$

Результат цієї операції визначений у деякому розширенні сорту ***Interval*.** Зворотна операція – операція додавання відрізків – визначена співвідношенням

$$[a,c] + +[c,b] = [a,b] \qquad \textbf{(5.44)}$$

Сорт VarSegment – сорт елементарних багатогранників простору розв'язків СЛН. Елемент цього сорту описується конструктором

$$S = [L(Y), x, G(Y)]$$

з семантикою $[L(Y) \le x \le G(Y)]$ де x – змінна, $Y = \{x_1, ... x_k\}$**,** $\mathrm{L(Y)}, \mathrm{G(Y)}$ – лінійні комбінації змінних сорту *Variable* з коефіцієнтами з *Coef*, тобто елементами сорту *LinComb(Y, Coef)***.** Елементи сорту *VarSegment* ми будемо називати *x-сегментами* або просто сегментами. Сорт *Interval* є частинним випадком сорту ***VarSegment*.**

Сорт Trapezoid – сорт елементарних трапецоїдів у просторі розв'язків СЛН. Для двовимірного простору $W(x, y)$ елемент цього сорту заданий конструкцією

$$T = [l,x,g].[L(x),y,Q(x)]$$

з семантикою $T = [l \le x \le g] \& [L(x) \le y \le Q(x)]$. Для n-вимірного простору $W(X)$, де $X = \{x_1,\ldots,x_n\}$, елемент цього сорту заданий конструкцією

$$T = [L_n(X_{n-1}) \le x_n, G_n(X_{n-1})] \ldots [L_2(x_1) \le x_2 \le G_2(x_1)].[L_1 \le x_1 \le G_1]$$

де $X_{k-1} = (x_1,\ldots,x_{k-1})$ з семантикою

$$T = (L_n(X_{n-1}) \le x_n \le G_n(X_{n-1})) \& \ldots \& (L_2(x_1) \le x_2 \le G_2(x_1)) \& (L_1 \le x_1 \le G_1) \ .$$

Точка «.» – відмітка конструктора сорту *Trapezoid*. Можливі й інші позначення:

$$T = S_x \cdot S_y, \ S_x = [l,x,g], \ S_y = [L,y,Q], \ T = S_l^g(x) \cdot S_L^Q(y).$$

Примітка. Трапецоїд – опукла фігура в n-вимірному просторі, послідовні проекції якої на підпростори меншої розмірності – також трапецоїди. Рівняння $x_n = L(X_{n-1}), \ x_n = G(X_{n-1})$ задають нижню й верхню «шапки» трапецоїда, якщо числову вісь O_{x_n} розглядати як вертикальну. Таким чином, якщо трапецоїд T задано формулою $T = S.T_1$, x_n –відрізок S визначає T за координатою x_n, а T_1 є проекцією T на $W(x_1,\ldots,x_{n-1})$. Точку – відмітку конструктора сорту *Trapezoid* можна інтерпретувати як операцію перетинання, якщо ввести у розгляд одиницю напівгрупи – нескінченний сегмент $I(x) = [Inf,x,+Inf]$. Тоді для $T = S_1(y).S_2(x)$, $T_1 = S_1(y).I(x)$, $T_2 = I(y).S_2(x)$ справедливе співвідношення $T = T_1 \& T_2$. Таким чином, алгебра трапецоїдів представлена у вигляді прямої границі зростаючої послідовності алгебр

$$TR(x_1) \subset TR(x_1,x_2) \subset \ldots \subset TR(x_1,\ldots,x_n) \subset \ldots$$

$$TR = \bigcup_{j=1}^{\infty} TR(X_j), \quad X_j = \{x_1,\ldots x_j\} \subset Variable$$

В термінології глави 3 *Trapezoid* – лінійне динамічне розширення алгебри *VarSegment*.

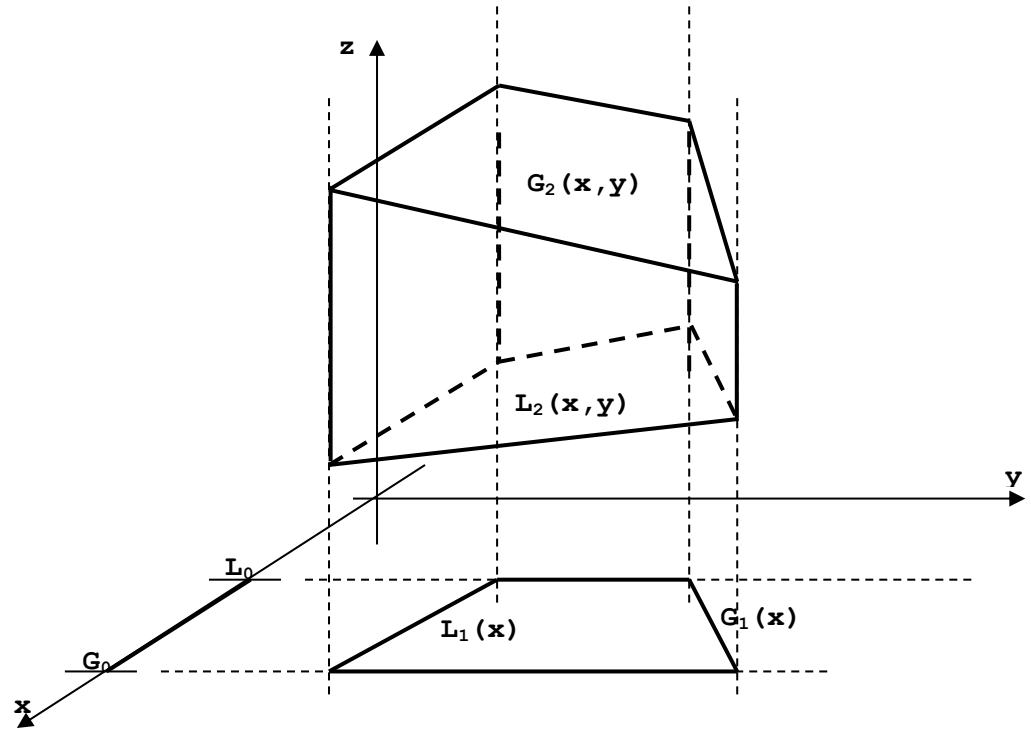


Рис. 5.8. Трапецоїд і його проекції в тривимірному просторі **W(x,y,z)**

Крім одиниці, в алгебрі *Trapezoid* є ще й нуль (анулятор) – пустий числовий проміжок. Нулем є елемент *VarSegment* виду $[a,x,b],\ a,b \in Coef$ **.** Таким чином, мають місце співвідношення

$$(a \in Coef)\&(b \in Coef)\&(a > b) \to [a,x,b] = O,\ \ S.O = O$$

Крім цього, властивості нульового елемента операції додавання сегментів має сегмент виду $[A,x,A]$. Більш точно, мають місце співвідношення

$$[L,x,G] ++ [G,x,G] = [L,x,G]\ \textbf{,}\quad [L,x,L] ++ [L,x,G] = [L,x,G]\ \textbf{.}$$

Copm ConPol **(**Convex Polygon**)** – сорт опуклих багатогранників у просторі розв'язків СЛН. В n-вимірному просторі елемент цього сорту (опуклий багатогранник) заданий конструкцією

$$P = T_1 ++ T_2 ++ \ldots ++ T_k$$

Примітка. Відмітка «++» конструктора сорту визначає розбиття опуклого багатогранника на трапецоїди, що перетинаються лише у підпросторі меншої розмірності – по границях. Лінійний порядок, зафіксований на множині змінних *Variable*, визначає послідовність проектувань опуклого багатогранника P на підпростори все меншої й меншої розмірності. Основна ідея такої конструкції навіяна наступною задачею: нехай G – багатогранна область n-вимірного простору, задана системою лінійних нерівностей, і необхідно обчислити (кратний) інтеграл $\int_G f(X)dX$. При переході

від кратного інтеграла до повторного необхідно визначити порядок інтегрування й границі інтегрування. Границі інтегрування мають бути аналітичними виразами, тобто лінійними комбінаціями змінних. Таким чином, шуканий кратний інтеграл треба представити у вигляді суми повторних інтегралів, у кожному з яких границі інтегрування визначають саме трапецоїди.

$$\int_G f(X)dX = \sum_j \int_{T_j} f(X)dX., \int_{T_j} f(X)dX = \int_{L_n}^{G_n} \int_{L_{n-1}}^{G_{n-1}} ... \int_{l_1}^{g_1} f(x_1,...,x_n)dx_1dx_2...dx_n$$

Зрозуміло, що розв'язок цієї задачі існує й, якщо встановити порядок інтегрування й зажадати мінімальності кількості областей розбиття, таке представлення області інтегрування єдине з точністю до комутативності.

Основна операція сорту *ConPol* – операція перетинання двох опуклих багатогранників. Ще одна операція – визначена частково операція додавання/розбиття. Сорт *ConPol* є лінійним динамічним розширенням сорту ***Trapezoid,*** тому ці операції визначаються через відповідні операції сорту ***Trapezoid.*** Сорт ***Trapezoid,*** у свою чергу, є розширенням сорту ***VarSegment,*** тому ці операції, у свою чергу, визначаються через відповідні операції сорту ***VarSegment.***

5.5.4. Операція перетинання на сорті ConPol

Ефективність алгоритму у цілому залежить від ефективності реалізації операції перетинання «&» на сорті *ConPol.* Операцію задано на сорті *ConPol* через її визначення на базових сортах *VarSegment* і ***Trapezoid.*** На сорті *VarSegment* перетинання визначено формулами

$$[L_1(X),y,G_1(X)] \& [L_2(X),y,G_2(X)] = [max(L_1,L_2),y,min(G_1,G_2)]$$

Результат операції має задовольняти нерівності $\max(L_1, L_2) \le \min(G_1, G_2)$.

На рис. 5.9 перелічені можливі варіанти взаємного розташування границь сегментів у випадку $n = 2$, при якому розглянуті багатогранники – суть трапеції. По–перше, обмежимся операцією перетинання трапеції з півплощиною $y \le G(X)$. Аналогічно розглядається перетин трапеції з півплощиною $y \ge L(X)$. По–друге, вважатимемо, що перетин півплощини і сегмента $S_1 = [L_1(X),y,G_1(X)]$ визначений на трапецоїді T підпростору $W(X_{n-1})$. Операція перетинання визначена на елементах $S_1.T$, $S.T$, причому $L_1(X) \underset{X \in T}{\le} G_1(X)$. При $n = 2$ трапецоїд T є відрізком числової осі $Ox = [L_0,x,G_0]$.

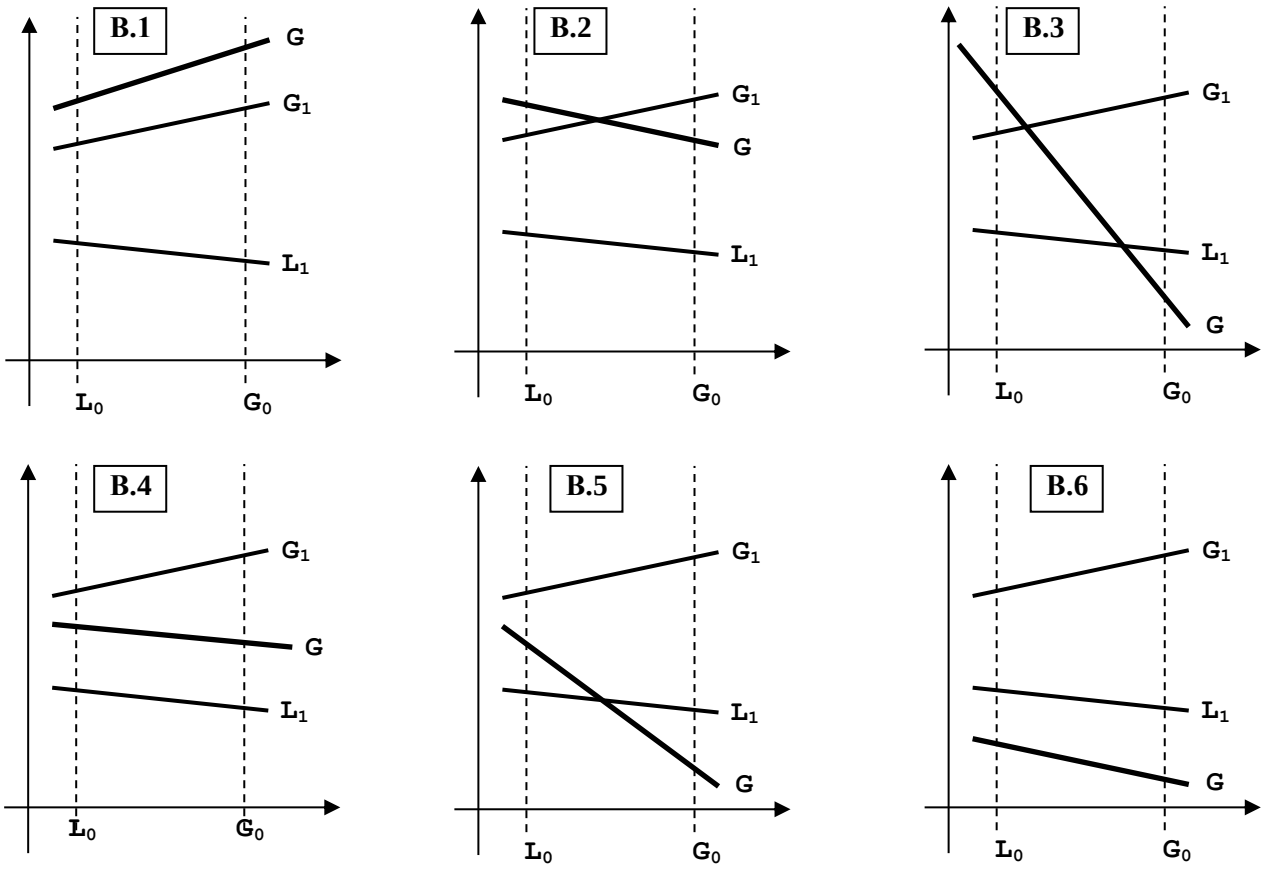


Рис. 5.9. Можливі варіанти перетинання трапецоїда напівплощиною

Варіант 1.

$G \geq_T G_1 \Rightarrow S_1.T \cap S = S_1.T.$

Варіант 2.

$(G \& G_1 \neq_T \varnothing) \& (G \geq_T L_1) \Rightarrow S_1.T \cap S = [L_1, y, G_1].T_1 + [L_1, y, G].T_2,$

$T_1 = \{X: G_1 \leq_T G\}, \; T_2 = T - T_1.$

Варіант 3.

$(G \& G_1 \neq_T \varnothing) \& (G \& L_1 \neq_T \varnothing) \Rightarrow S_1.T \cap S = [L_1, y, G_1].T_1 + [L_1, y, G].T_2$

$T_1 = \{X: G_1 \leq_T G\}, \; T_2 = \{X: G \leq_T G_1\}$

Варіант 4.

$(G \leq_T L_1) \& (G \leq_T G_1) \Rightarrow S_1.T \cap S = [G, y, L_1].T$

Варіант 5.

$(G \leq_T L_1) \& (G \& L_1 \neq_T \varnothing) \Rightarrow S_1.T \cap S = [L_1, y, G].T_1,$

$T_1 = \{X: L_1 \leq_T G\}$

Варіант 6.

$G \leq_T G_1 \Rightarrow S_1.T \; \& \; S = \varnothing$

Основним джерелом неефективності алгоритму, описаного співвідношеннями цих варіантів, є наявність двох доданків у правих частинах співвідношень і необхідність обчислень нових трапецоїдів–проекцій T_1, T_2. У варіанті 6 кількість трапецоїдів у перетинанні зменшується, у варіантах 1, 4 кількість трапецоїдів не змінюється, причому проекції залишаються незмінними, у варіанті 5 кількість трапецоїдів не змінюється, але проекція

змінюється. Найгірші випадки – варіанти 2, 3, у яких кількість трапецоїдів збільшуються й проекції змінюються.

Алгоритм розпізнавання варіанта. Усі співвідношення, наведені у варіантах рис 6.9, є умовними. Умови визначені в термінах відношення часткового порядку. $L \le_T G$ означає, що на області T виконується нерівність $L(X) \le G(X)$. У свою чергу, це означає, що $y_{\max} = \max\limits_{X \in T}(L(X) - G(X)) \le 0$. Функція $y = L(X) - G(X)$ є лінійною, а трапецоїд T є багатогранною областю. Звідки випливає, що максимум $y_{\max}$ досягається в одній з вершин T. Нарешті, форма представлення трапецоїда у вигляді послідовності його проекцій на підпростори все меншої розмірності дозволяє обчислити максимум за час $O(n^2)$. Покажемо це.

1. Приведемо трапецоїд $T_n = S_1 S_2 \cdot \ldots \cdot S_n$ до виду $S_i = [0, x_i, g_i(X_{i-1})]$ перетворенням

$$S_i = [l_i(X_{i-1}), x_i, g_i(X_{i-1})] \Rightarrow [0, x_i - l_i(X_{i-1}), g_i(X_{i-1}) - l_i(X_{i-1})]$$

і наступною заміною змінних $x_i = x_i - l_i(x_{i-1})$. Ці перетворення виконуються на трапецоїді за час $O(n^2)$.

2. Розглянемо наступну задачу лінійного програмування (ЗЛП): Знайти $\max F$ лінійної функції $F = c_1 x_1 + \ldots + c_{n-1} x_{n-1} + c_n x_n$ на приведеному трапецоїді T_n. Оскільки розв'язок ЗЛП досягається на границях трапецоїда T_n, розглянемо два випадки:

2.1 $x_n = 0$. Тоді ЗЛП зводиться до задачі розмірності $n-1$: на трапецоїді $T_{n-1} = S_1 S_2 \cdot \ldots \cdot S_{n-1}$ знайти $\max F_1$, $F_1 = c_1 x_1 + \ldots + c_{n-1} x_{n-1}$.

2.2. $x_n = g_n(X_{n-1}) \ldots$ Тоді ЗЛП зводиться до наступної ЗЛП розмірності $n-1$: на трапецоіді $T_{n-1} = S_1 S_2 \cdot \ldots \cdot S_{n-1}$ знайти $\max F_2$, $F_2 = c_1 x_1 + \ldots + c_{n-1} x_{n-1} + c_n x_n(X_{n-1})$.

Таким чином,

$$\max F = \max(\max F_1, \max F_2) \qquad \textbf{(5.45)}$$

Оскільки на T_{n-1} $g_n(X_{n-1}) \ge 0$, маємо:

a) $c_n < 0$. $\max(\max F_1, \max F_2) = \max(\max F_1, F_1 - g) = \max F_1, g \ge 0$.

b) $c_n \ge 0$. $\max(\max F_1, \max F_2) = \max(\max F_1, F_2) = \max F_2, g \ge 0$.

Отже, ЗЛП розміру n на трапецоїді зводиться до задачі розміру $n-1$ на трапецоїді за час $O(n)$.

Ці співвідношення треба доповнити частинними випадками, які визначаються тим, що сорт *ExtCoef* є розширенням алгебри *Coef* елементами $Inf, +Inf$.

Якщо сегмент S має вигляд $S=[-Inf, x_n, G(X_{n-1})]$, його не можна привести до виду, визначеного у п.1 Однак, при $c_n < 0$ $\max F = +Inf$. При $c_n > 0$ по суті має місце випадок п.1 і ЗЛП розв'язується формулою (**5.45**). Аналогічно розглядається випадок, коли необмежений сегмент задається формулою $S=[L(X_{n-1})x_n, +Inf]$. Для реалізації обчислень у цих випадках визначається сорт *ExtCoef***.** Обчислення в алгебрі *ExtCoef* визначаються правилами

```
a > 0 -> a*(+Inf) = +Inf,
a > 0 -> a*(-Inf) = -Inf,
a < 0 -> a*(+Inf) = -Inf,
a < 0 -> a*(-Inf) = +Inf,
a + +Inf = +Inf,
a + -Inf = -Inf,
a - +Inf = -Inf,
a - -Inf = +Inf.
```

Отже, ЗЛП розміру n розв'язується на трапецоїді за час $O(n^2)$.

Примітка. Описаний алгоритм розв'язує ЗЛП на трапецоїді. Нижче ми побачимо (**5.46**), (**5.47**), що багатогранна область представляється у вигляді розбиття на трапецоїди. Тому ЗЛП можна розв'язати цим методом на довільній опуклій багатогранній області.

5.5.5. Канонічні форми сорту ConPol

Як ми вже відзначали, опуклий багатогранник – елемент сорту *ConPol* можна представити у вигляді розбивки його на трапецоїди. Таким чином,

$$\mathrm{P} = \mathrm{T}_1 + + \mathrm{T}_2 + + \ldots + + \mathrm{T}_\mathrm{k} \qquad (\mathbf{5.46})$$

На множині трапецоїдів природно може бути задане відношення лінійного порядку, індуковане відношеннями порядку на сортах *Coef, Variable* і розширене спочатку на сорт *Trapezoid*, а потім і на сорт *ConPol***.** Таким чином, можна вважати, що

$$\mathrm{T}_1 > \mathrm{T}_2 > \ldots > \mathrm{T}_\mathrm{k}\ , \quad \mathrm{T}_\mathrm{j} = \mathrm{S}_{\mathrm{jn}}.\mathrm{S}_{\mathrm{jn\text{-}1}}.\ldots.\mathrm{S}_{\mathrm{j1}} \qquad (\mathbf{5.47})$$

Застосовуючи співвідношення $S.0 = 0,\ S.I = S,\ T + +0 = T,\ 0 + +T = T$, можна позбутися від «зайвих» доданків і співмножників. Тоді елемент P представлено в канонічній формі

$$P = \sum_{j=1}^{k} \prod_{i=1}^{n} S_{ji} \qquad (\mathbf{5.48})$$

Наприклад, $P = S_{13}.S_{12}.S_{11} + + S_{23}.S_{22}.S_{21} + + S_{33}.S_{32}.S_{31} + + S_{43}.S_{42}.S_{41}$.

Отже, сорт *ConPol* є лінійним динамічним розширенням сорту ***Trapezoid*.** Предикатні змінні S_{jk} інтерпретовані в алгебрі ***VarSegment*.**

Можна показати, що множина виразів такого виду задовольняє більшості аксіом алгебри булевих функцій в сигнатурі операцій–відміток $<++,.,O,I>$**.** Тому багатогранники можна називати поліномами, трапецоїди – мономами, а сегменти – змінними.

Примітка. Перестановка співмножників у добутках $S_{jn}.S_{jn-1}.\ldots.S_{j1}$ – це задача зміни порядку інтегрування. Ця задача становить самостійний інтерес, причому не тільки для інтегралів по опуклих багатогранниках, але й, наприклад, для областей, заданих системами нелінійних опуклих нерівностей.

Оскільки в конструктивній алгебрі *ConPol* має місце закон дистрибутивності, поряд з поданням полінома P у вигляді суми мономів T_j, можна визначити ще одну канонічну форму – так зване рекурсивне представлення P**.**

Індексом (розмірності) змінної – елемента $S = [L, y, G]$ назвемо порядковий номер змінної y у сорті ***Variable*.** Індексом елемента (монома) T назвемо індекс його старшого елемента. Нарешті, індексом полінома є індекс його старшого монома.

Нехай в (**5.46**) кілька доданків мають вигляд $S.T_1, \ldots S.T_k$, де S – змінна максимального індексу. Тоді S можна винести за дужки. Здійснивши це перетворення для всіх змінних S_j максимального індексу, з (**5.48**) отримаємо:

$$P = S_1.P_1 + + S_2.P_2 + \ldots + S_k.P_k \qquad (\mathbf{5.49})$$

У (**5.49**) індекси поліномів P_j менше, ніж індекси змінних S_j. Застосовуючи рекурсивно це перетворення до всіх поліномів все меншого індексу, отримаємо рекурсивну канонічну форму елемента алгебри *ConPol*. Наступне перетворення, засноване на властивості додавання (**5.44**), ми будемо називати зведенням подібних.

Нехай $P = S_1.P_1 + + S_2.P_1$**,** причому $S_1 = [A, x, C]$, $S_2 = [C, x, B]$**.** Тоді $P = [A, x, B].P_1$. Таким чином, зведення подібних визначається правилом

$$[A, x, C].T + + [C, x, B].T = [A, x, B].T \qquad (\mathbf{5.50})$$

Рекурсивною нормальною формою (РНФ) полігона – елемента алгебри *ConPol* будемо називати форму (**5.49**), у якій зведені всі подібні за правилом (**5.50**). Зрозуміло, що РНФ більш економно представляє полігон, ніж поліноміальна нормальна форма (ПНФ) (**5.48**).

Відзначимо, що можна розглядати й інші канонічні форми подання полігонів. Так, для побудови базисної системи векторів полігону ми представимо полігони як перетинання «нижніх» і «верхніх» шапок.

5.5.6. Ізоморфізми представлень алгебри СЛН

Алгоритм розв'язання СЛН починає свою роботу з перетворень кожної лінійної нерівності системи в елементарний сегмент (атом). Відповідні правила мають вигляд: $\mu(y \le G) = [-Inf, y, G]$, $\mu(L \le y) = [L, y, +Inf]$. Ці перетворення визначають ізоморфізм алгебри $\mu : A_{Init} \to A_{Constr}$. В реалізації має сенс об'єднати їх з обчисленнями *x*-розв'язуваної форми ЛН.

По суті, полігон P, представлений у РНФ, описує розв'язок СЛН. Однак, якщо необхідно представити тільки оболонку розв'язку в n–вимірному просторі, ігноруючи проекції в простори меншої розмірності, це можна зробити за допомогою зворотного ізоморфізму $\mu^{-1} : A_{Constr} \to A_{Init}$. Відповідне основне правило має вигляд $\mu^{-1}([L, x, G).P + +Q] = (L \le x) \& (x \le G) \& \mu^{-1}(Q)$.

5.5.7. Управління і складність обчислень

Очевидно, найбільш простий підхід до управління обчисленнями полягає в послідовному додаванні до вже побудованої РНФ нової нерівності. Як ми показали, алгоритм переобчислює нерівності, що визначають трапецоїд, за час $O(n^2)$. Нехай $C(m,n)$ – кількість трапецоїдів (мономів) у поточному представленні (5.22). Тоді переобчислення полінома потребує $O(n^2)C(m,n)$ кроків. Оскільки алгоритм додавання нерівності повторюється m разів, $T(m,n) = O(mn^2 C(m,n))$. Неважко показати, що в розбивці багатогранної області P на трапецоїди кожний трапецоїд можна асоціювати принаймні з однією (своєю) вершиною P. Інакше трапецоїд можна було б «розширити», збільшивши один з його сегментів. Тому кількість трапецоїдів в P не перевершує кількості V його вершин (0–граней) (враховуються і нескінченно віддалені вершини). Однак трапецоїд визначається $2n$ нерівностями і містить 2^n. вершин. Тому оцінку методу $T(m,n) = O(mn^2 V(m,n))$, у якій $V(m,n)$– кількість вершин P, є експоненціальною. Специфікації сортів алгебри A_{Constr} наведено у додатку 5.

5.5.8. Обчислення множини базисних векторів

Нехай полігон P в n-вимірному просторі заданий канонічною формою (5.22), тобто розбивкою на трапецоїди.

$$P = T_1 + + T_2 + + \ldots + + T_k,\ \mathrm{T_i} = \mathrm{S_{i1}}.\mathrm{S_{i2}}.\ldots.\mathrm{S_{in}},\ \mathrm{S_{ij}} = [\mathrm{L_{ij}}, \mathrm{x_j}, \mathrm{G_{ij}}]. \qquad (\mathbf{5.51})$$

Обчислення вершин (0–граней) полігона – одна з основних задач теорії систем лінійних нерівностей. Легко бачити, що всі шукані вершини – суть вершини трапецоїдів $\mathrm{T_1}, \ldots, \mathrm{T_k}$. Зворотне, однак, невірно. Серед вершин трапецоїдів багато зайвих. Причина полягає у тому, що розбивка на трапецоїди враховує і нижні, і верхні грані спільно (див. рис. **5.10**).

На рис. 5.10 видно, що вершина B_1 нижньої грані P, є вершиною нижньої грані трапецоїда T_2, однак відповідна їй точка A_1 верхньої грані не є вершиною, а належить стороні P.

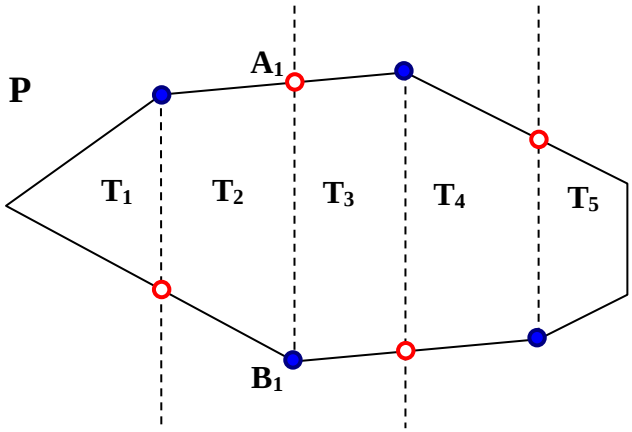


Рис. 5.10. Розбивка полігона на трапецоїди по верхній і нижній шапках

Для обчислення вершин полігона P спочатку перетворимо канонічну форму (**5.51**) у такий спосіб:

1. Для кожного трапецоїда $T_i = S_{i1}.T_i^{(n-1)},\ S_{i1} = [L_{i1}, x_n, G_{i1}]$ виділимо його нижню й верхню "шапки":

$$T_i^+ += S_{i1}^+ + .T_i^{(n-1)},\ T_i^- = S_{i1}^-.T_i^{(n-1)},$$

$$S_{i1}^+ += [-Inf,\ x_n,\ G_{i1}] \stackrel{df}{=} (x_n,\ G_{i1}],\qquad S_{i1}^- = [L_{i1}, x_n, + Inf] \stackrel{df}{=} [L_{i1}, x_n).$$

Канонічну форму перетворимо до виду

$$P = (\ T_1^+ + + T_2^+ + + \ldots + + T_k^+,\ T_1^- + + T_2^- + + \ldots + + T_k^-\ ),$$

$$s_{i1}^+.T_1 + + s_{i1}^+.T_2 + + \ldots + + s_{i1}^+.Tl = s_{i1}^+.(T_1 + + T_2 + + \ldots + + T_l\ )$$

і зведемо подібні у дужках. А саме: P запишемо у вигляді $P = (P^+, P^-)$**.** У поліномах P^+, P^- тепер можливе зведення подібних. Використовуючи властивість дистрибутивності, винесемо за дужки одночлен s_{i1}^+ для всіх доданків виду $s_{i1}^+.\mathrm{T_k}$:

$$[L, x, H].T + + [H, x, G].T = [L, x, G].T$$

В результаті отримаємо представлення $(n-1)$ – грані з рівнянням $x_n = G_{i1}$, заданою s_{i1} і обмеженої $(n-1)$-вимірним полігоном $T_1 + + \ldots + + T_m$ **:**

$$s_{i1}^+.(T_1 + + \ldots + + T_m\ )$$

Отже, полігон P можна представити у вигляді

$$P = (P^+, P^-),$$

$$P^+ = S_1^+.P_1^+ + + \ldots + + S_l^+.P_l^+,$$

$$P^- = S_1^-.P_1^- + + \ldots + + S_r^-.P_r^-,$$

де S_i^+, S_j^- – рівняння $(n-1)$-граней верхньої й нижньої шапок полігона P, а P_i^+, P_j^- – $(n-1)$-мірні полігони, що обмежують ці грані. Застосовуючи рекурсивно зверху вниз перетворення, визначені вище, до P_i^+, P_j^-, отримаємо ще одну канонічну форму P, кожна i-грань якої представлена своїми вершинами.

Перетворення ілюструються рис. 5.11.1 – 5.11.3:

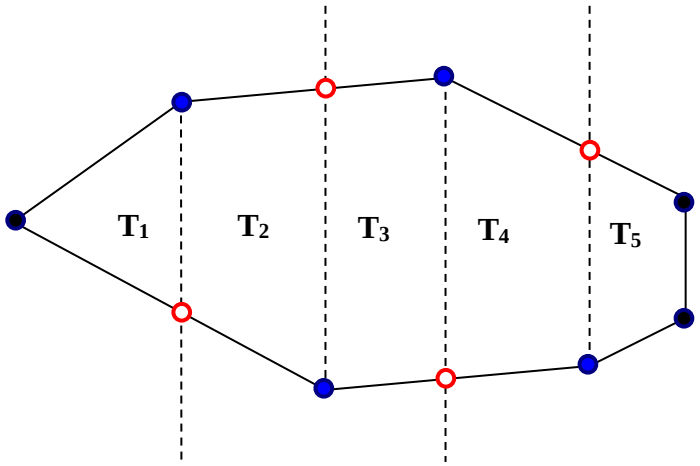


Рис. 5.11.1. Розбиття полігону на трапецоїди.

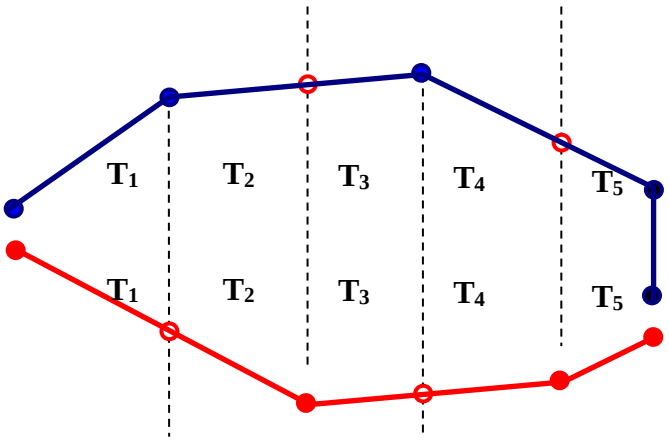


Рис. 5.11.2. Відокремлення верхніх і нижніх «шапок».

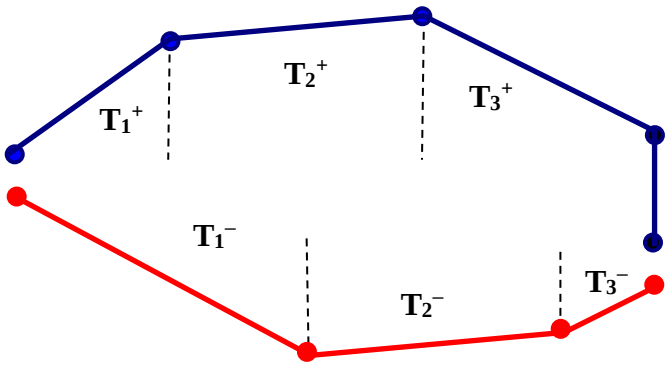


Рис. 5.11.3. Перетворення полігона до третьої канонічної форми

Продовжимо розгляд. Позначимо на рис. 5.12 систему координат.

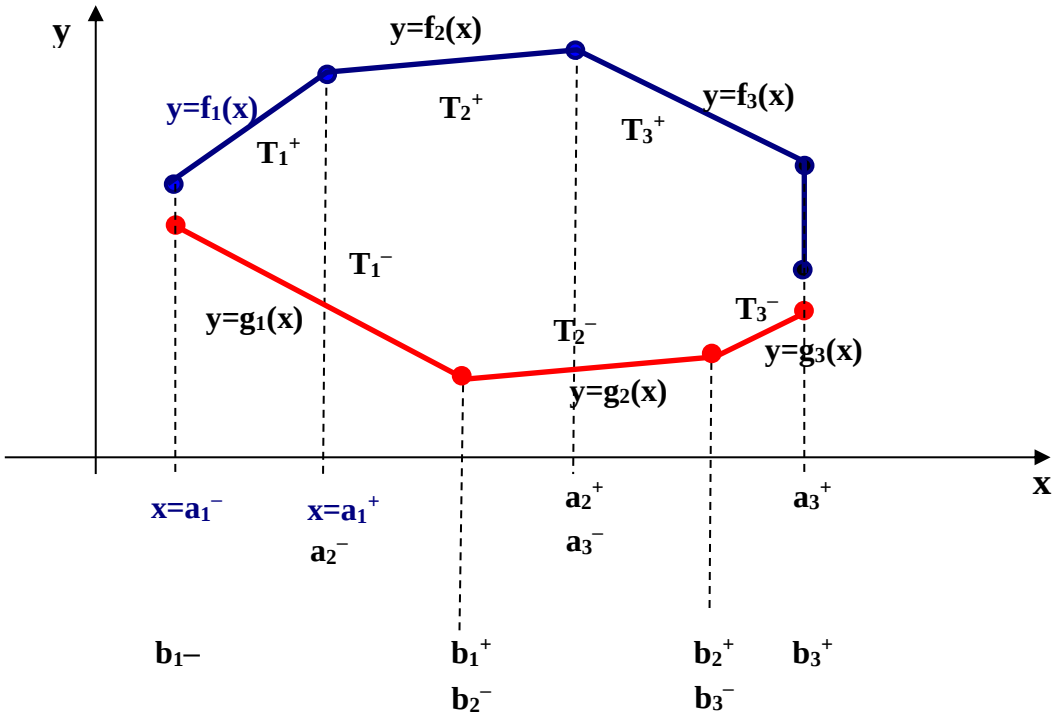


Рис. 5.12. Ілюстрація алгоритму обчислення вершин полігона.

Тоді

$$P^+ = y = f_1(x).(x = a_1^-, x = a_1^+) + + y = f_2(x).(x = a_2^-, x = a_2^+) + + y =$$
$$= f_3(x).(x = a_3^-, x = a_3^+) + + x = a_3^+$$

$$P^- = y = g_1(x).(x = b_1^-, x = b_1^+) + + y = g_2(x).(x = b_2^-, x = b_2^+) + + y =$$
$$= g_3(x).(x = b_3^-, x = b_3^+)$$

Обчислення базисних вершин. Очевидний і простий алгоритм обчислення базисних вершин полігона (базисних векторів) полягає у наступному:

1. Обчислення здійснюються знизу–вгору. При цьому обчислюються базиси *k*-граней по k від 1 *до* n.

2. Якщо обчислено базис $V^{(k)}(P)=\{v_1,...,v_l\}$ і рівняння шапки s має вигляд $x_{k+1}=f(x_1,...,x_k)$, то базис $k+1$-грані $S.P$ дорівнює

$$V^{(k+1)}(S.P)=\{(v_1,f(v_1)),...,(v_l,f(v_l))\} \tag{5.52}$$

3. Якщо обчислені базиси $V^{(k)}(P)=\{v_1,...,v_l\}$, $V^{(k)}(Q)=\{u_1,...,u_r\}$ k-полігонів P,Q, то базис $V^{(k)}(P++Q)$ дорівнює

$$V^{(k)}(P++Q)=V^{(k)}(P)\cup V^{(k)}(Q) \tag{5.53}$$

4. Якщо обчислені базиси $V^{(k)}(P^+)=\{v_1,...,v_l\}$, $V^{(k)}(P^-)=\{u_1,...,u_r\}$ k-полігонів P^+,P^-, то базис $V^{(k)}(P^+,P^-)$ дорівнює

$$V^{(k)}(P^+,P^-)=V^{(k)}(P^+)\cup V^{(k)}(P^-) \tag{5.54}$$

Базис рекурсивних обчислень очевидний: $V^{(1)}(x_1=a)=(a)$.

Суміжні грані обчислюються аналогічно: якщо обчислені базиси $V^{(k)}(P)=\{v_1,...,v_l\}$, $V^{(k)}(Q)=\{u_1,...,u_r\}$ k–полігонів P,Q, то базис $V^{(k)}(P\cap Q)$ дорівнює

$$V^{(k)}(P\cap Q)=V^{(k)}(P)\cap V^{(k)}(Q) \tag{5.55}$$

Відзначимо, що співвідношення (5.52)-(5.55) по суті визначають ізоморфізм алгебри полігонів в алгебру множин векторів.

5.6. Висновки

1. Для багатьох задач, які виникають при розробленні систем комп'ютерної математики, основними алгоритмами комп'ютерної алгебри є алгоритми побудови базисів Гребнера, відділення та уточнення дійсних коренів поліномів, побудови примітивного елемента поля розкладу полінома.
2. Алгоритми цих задач можуть бути неефективними в силу того, що неефективними є алгоритми побудови базисів Гребнера та примітивного елемента поля розкладу полінома. Зокрема, прикладом є алгоритм побудови групи Галуа полінома.
3. Алгоритм дослідження функції дійсного аргументу має сенс реалізувати для функцій невеликої кількості аргументів.
4. Новий алгоритм розв'язання систем лінійних нерівностей, оснований на алгебраїчному підході, має велике прикладне значення для багатьох виробничих математичних систем. Зокрема, одна з його позитивних рис – можливість ефективної перебудови канонічних форм при додаванні до системи нерівностей нової нерівності.

Глава 6. Методи комп'ютерної алгебри в задачах статичного аналізу програм.

6.1. Моделі програм

В теорії програмування об'єкт дослідження, як правило, представлений математичною моделлю програми. В [85] це операторні схеми програм, схеми програм Янова. В теорії статичного аналізу та перетворення програм використовуються або графові [86-88], або алгебраїчні моделі програм [92-94]. До графових моделей відносяться, наприклад, $U-Y$ схеми програм [86, 91], переходові системи (transition systems) [95]. Алгебраїчними моделями є алгоритмічні алгебри В.М.Глушкова, терми програмних логік.

Програмні системи, призначені для розв'язання задач аналізу та перетворення програм, мають оперувати текстами програм, написаними однією з мов програмування високого рівня. Отже, такі системи містять спеціальні програмні модулі трансляції вихідного тексту програми в її математичну модель. В методах перетворення та синтезу програм важливою є задача зворотної трансляції – перетворення математичної моделі у вихідний код програми.

код програми $\Leftrightarrow$ *математична модель програми*

6.1.1. Вибір математичної моделі

У теоретичних роботах, присвячених задачам автоматичної генерації інваріантів програм використовуються як графові моделі програм, так і алгебраїчні моделі програм.

Загальними поняттями для цих моделей є:

- поняття пам'яті X програми P як скінченної множини (вектора) змінних $X=(x_1,...,x_n)$;
- поняття алгебри даних D як області визначення всіх змінних (пам'яті) програми. Алгебра даних визначає множину операцій над даними. Взагалі кажучи, D є багатосортною алгеброю. Одним із сортів D є *Boolean*. У наших задачах, крім *Boolean,* визначений ще тільки один сорт – сорт, над яким за допомогою операторів присвоювання визначені обчислення;
- поняття стану $\bar{d}$ пам'яті програми P як вектора $\bar{d}=(d_1,...,d_n)$. Якщо пам'ять програми перебуває в стані $\bar{d}$, це означає, що $x_1=d_1,...,x_n=d_n$. Таким чином, ми визначаємо простір станів пам'яті як множину D^X;
- оператори присвоєння інтерпретуються як векторні. Оператор $y=(x_1:=F_1(X),...,x_n:=F_n(X))$

  перетворює простір станів пам'яті: $y: D^X \to D^X$. Це означає, що компоненти оператора присвоювання виконуються одночасно (паралельно). Множину операторів присвоєння позначимо через Y.

Відзначимо, що в деяких задачах аналізу програм варто розглядати послідовне виконання компонентів;

- умови перетворюють терм F алгебри D в (*True, False*). Множину умов позначимо через U.

6.1.2. Графові моделі програм. $U-Y$ - програми

$U-Y$ – схемою програми над пам'яттю ($U-Y$-програмою) називають ініціальний зв'язний граф P з множиною вершин S, виділеною ініціальною вершиною $s_0 \in S$, підмножиною $S^* \subseteq S$ заключних вершин (станів), множиною дуг E, причому дуги відмічені парами (u, y) – (умова-оператор).

$$P = < S, E, s_0, S^*, \delta >,\ \delta : E \to (U, Y)$$

Множина S вершин графа P – суть множина контрольних точок програми. Перехід від однієї точки до іншої супроводжується виконанням оператора y, якщо u у поточному стані пам'яті програми приймає значення *True*.

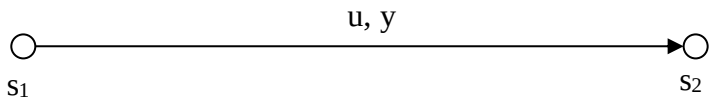


Рис. 6.1. Перехід в $U-Y$ програмі.

Виконання програми P починається в стані s_0, здійснюється недетермінованим чином й завершується в одному із заключних станів S^*.

6.1.3. Алгебраїчні моделі програм. Алгоритмічна алгебра В.М.Глушкова

Алгебраїчні моделі імперативних структурованих програм використовують представлення програми термом (формулою) спеціальної програмної алгебри, операціями якої є оператори управління: послідовне виконання, розгалуження, повторення. Наприклад:

- $P;Q$ – послідовне виконання програм P,Q ;
- $P \underset{u}{\vee} Q$ – розгалуження програм P,Q за умовою u;
- $\{P\}_u$ – повторення програми P з передумовою u;
- $\{P\}_u$ – повторення з програми P постумовою u.

Приклад 6.1. (Програма та її алгебраїчна модель)

Розглянемо, як приклад програми та її алгебраїчної моделі, розширений алгоритм Евкліда, що знаходить $b = GCD(x, y)$ і такі числа c, d, що $cx + dy = b$.

```
Procedure ExtGCD(x, y: Integer; var b, c, d: Integer);
Var
  a, u, v, q, r : Integer;
Begin
  a:=x; b:=y;
  c:=0; d:=1;
  u:=1; v:=0;
  q:=0; r:=x
  While r >= b do begin
    r:=r-b;
    q:=q+1
  End;
  While r<>0 do begin
    a:=b; b:=r;
    c:=u-q*c; d:=v-q*d;
    u:=c; v:=d;
    q:=0; r:=b
    While r >= b do begin
      r:=r-b;
      q:=q+1
    End
  End
End;
```

Для задач аналізу програм з тексту програми потрібно сформувати:

$E = (x, y, b, c, d)$ – перелік вхідних параметрів програми;

$X = (a, b, c, d, q, r, u, v, x, y)$ – пам'ять програми;

$U = (u_1, u_2)$ – перелік умов програми;

$u_1 = (r >= b)$; $u_2 = (r <> 0)$;

$Y = (y_1, y_2, y_3)$ – перелік операторів присвоювання програми;

$y_1 = (\text{a} := \text{x}, \text{b} := \text{y}, \text{c} := 0, \text{d} := 1, \text{u} := 1, \text{v} := 0, \text{q} := 0, \text{r} := \text{x})$;

$y_2 = (\text{r} := \text{r} - \text{b},\ \text{q} := \text{q} + 1)$;

$y_3 = (\text{a} := \text{b}, \text{b} := \text{r}, \text{c} := \text{u} - \text{q} * \text{c}, \text{d} := \text{v} - \text{q} * \text{d}, \text{u} := \text{c}, \text{v} := \text{d},\ \text{q} := 0, \text{r} := \text{b})$.

Алгебраїчна модель програми тепер визначається формулою:

$$P = y_1; \underset{u_1}{\{}\ y_2\}; \underset{u_2}{\{}\ y_3; \underset{u_1}{\{}\ y_2\}\}.$$

6.2. Задача автоматичної генерації інваріантів програм

6.2.1. Інваріантні рівності та інваріанти програм

Задача автоматичної генерації програмних інваріантів є однією з найважливіших задач статичного аналізу програм. Вона має довголітню історію, що починається з робіт Флойда та Хоара [89, 90]. Головною метою цієї гдави є розгляд задач теорії програмних інваріантів [91-94], [95-100]. Розглянемо елементарний приклад.

Приклад 6.2. Програма обчислення максимума двох чисел $M = max(a,b)$ Представимо алгоритм обчислення блок-схемою з контрольними точками C_0 - C^*. (рис.6.2.). Визначимо логічну умову, що має виконуватися в результаті виконання програми, тобто в контрольній точці C^* (постумову).

// Постумова $U^* = ((M = a) \vee M = b)) \& ((M >= a) \& (M >= b))$.

Аналіз програми полягає в обчисленні логічних умов у кожній з контрольних точок C_0 - C^* (аналіз потоку обчислень) та обчисленні інваріанта – логічної умови I в контрольній точці C^*. Доведення правильності (верифікація) полягає у доведенні $I \to U^*$. Обчислення I є задачею статичного аналізу програм. Доведення $I \to U^*$ – це задача математичної логіки.

Розглянемо клас програм, алгеброю даних D яких є конструктивне поле F. Через X позначимо множину змінних $\{x_1, x_2, ..., x_n\}$ програми P.

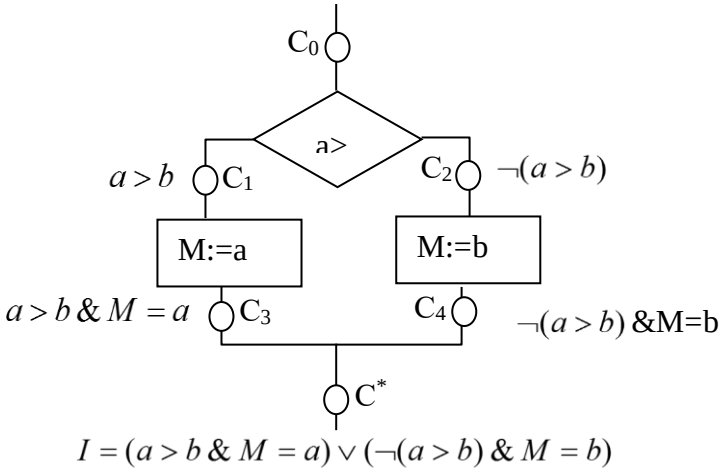


Рис. 6.2. Обчислення інваріанта програми $M = max(a,b)$

Означення 6.1. Поліном $g(X) \in F(X)$ називається поліноміальним програмним інваріантом програми P, якщо для довільного початкового стану пам'яті $a = (a_1,...,a_n)$, коли програма P завершила виконання, у заключному стані пам'яті $b = (b_1,...,b_n)$ виконується рівність $g(b) = 0$. Рівність $g(X) = 0$ називається інваріантною рівністю програми P.

Приклад 6.3. Інваріант та інваріантна рівність програми обчислення неповної частки q та остачі r від ділення a на b.

```
Procedure ModDiv(a, b: Integer; var q, r: Integer);
begin
  r := b;
  q := 0;
  While r >= b do begin
    r := r - a;
    q := q + 1
  end
  {Інваріантна рівність: a = b*q + r, Інваріант: a - b*q - r}
end;
```

Таким чином, поліноміальні інваріантні рівності мають вигляд $l(X) = r(X)$, де $l(X), r(X) \in F[X]$. Інваріант, що відповідає цій рівності, має вигляд $l(X) - r(X)$.

Для подальшого нам знадобляться деякі означення та факти теорії поліноміальних ідеалів.

Нехай F – конструктивне поле. Поле $\overline{F}$ називається алгебраїчним замиканням F, якщо $F \subset \overline{F}$ та для буль-якого кореня α будь-якого полінома $f(x) \in F[x]$ $\alpha \in \overline{F}$. Таким чином, поле $\overline{F}$ містить усі корені усіх поліномів з коефіцієртами з F.

Нехай I – ідеал кільця поліномів $F[X]$ і вектор $\overline{\alpha} = (\alpha_1,...,\alpha_n), \alpha_i \in \overline{F}$. Вектор $\overline{\alpha}$ називається коренем $f(X) \in F[X]$, якщо $f(\overline{\alpha}) = 0$. Многовидом ідеала $I \subseteq F[X]$ називається множина $M_I = \{\overline{\alpha} : \forall f \in I(f(\overline{\alpha}) = 0)\}$. Тобто, многовид M_I є множиною усіх коренів, спільних для усіх поліномів-елементів I.

Нехай $M \subseteq \overline{F}$. Очевидно, що $I_M = \{f : \forall \overline{\alpha} \in M(f(\overline{\alpha}) = 0)\}$ є ідеалом кільця $F[X]$. Нехай I – деякий ідеал кільця $F[X]$. Тоді можна визначити ідеал I_{M_I}. Очевидно, що $I \subseteq I_{M_I}$. Якщо $I = I_{M_I}$, ідеал I наивають ідеалом свого многовиду.

Ідеал I кільця R називається простим, якщо з $f \cdot g \in I$ слідує $f \in I$ або $g \in I$.

Ідеал $I \subseteq R$ називається радикальним, якщо з $f^k \in I$ для деякого натурального k слідує $f \in I$.

Поліноміальний ідеал I є ідеалом свого многовиду тоді і тільли тоді, коли I є радикальним ідеалом.

Будь-який радикальний ідеал $I \subseteq F[X]$ можна представити у вигляді перетину скінченного числа простих ідеалів: $I = J_1 \cap J_2 \cap ... \cap J_k$.

Зауважимо, що множина поліноміальних інваріантів програми P є ідеалом свого многовиду, і отже – радикальним ідеалом кільця $F[X]$, який ми будемо позначати через I_P.

Повна постановка задачі генерації поліноміальних інваріантів полягає в наступному:

Нехай I_P – ідеал поліноміальних інваріантів програми P. Треба представити I_P у вигляді перетину простих ідеалів $I_P = J_1 \cap J_2 \cap ... \cap J_l$ і побудувати базиси Гребнера ідеалів $J_1, J_2, ..., J_l$. Однак, цю задачу ще не розв'язано. Основними результатами теорії поліноміальних інваріантів є наступні теореми та алгоритми.

6.2.2. Метод генерації поліномальних інваріантів обмеженого ступеня.

Розглянемо множину елементів $I_P^{(M)} = \{g \in I_P \mid \deg g \le M\}$, тобто множину усіх поліноміальних поліномів, ступені яких обмежені константою M. Очевидно, що ця множина утворює скінченновимірний підпростір векторного простору I_P: $I_P^{(M)} \subset I_P \subset F[X]$.

Теорема 6.1. Якщо $ch(F) = 0$ і мова умов L містить предикати рівності та заперечення рівності (=, ≠), проблема побудови базису I_P є алгоритмічно не розв'язуваною навіть для класу програм, усі оператори присвоєння яких є лінійними перетвореннями.

Доведення викладено у [93]. Тому у подальшому ми будемо ігнорувати умови в програмах, вважаючи, що програми є абсолютно недетермінованими. Зауважимо, однак, що в [95] показано, що умови типу заперечення рівності можна враховувати.

Ігнорування умов спрощує алгебраїчну модель програми до регулярного виразу в сигнатурі $P \cdot Q,\ P \vee Q,\ \{P\}$. Основний результат сформульовано у наступній теоремі:

Теорема 6.2. Нехай P – програма, інтерпретована над поліноміальним кільцем $F[X]$, I_P – ідеал інваріантів P і $I_P^{(M)} = \{g \mid g \in I_P,\ \deg\ g \le M\}$.

Тоді:

a) Проблема приналежності $g \in I_P$ є алгоритмічно розв'язуваною;

b) Проблема побудови базису векторного простору $I_P^{(M)}$ є алгоритмічно розв'язуваною.

Доведення. (Індукція по структурі регулярного виразу, що представляє програму)

Нехай $y_1,...,y_m$ – перелік операторів присвоєння зі словника програми P, $y_i = (X := F_i(X))$.

Доведення будемо вести індукцією по структурі регулярного виразу цієї програми в сигнатурі (6), де оператори присвоєння $\boldsymbol{y_1,...,y_m}$ – по суті є атомами.

Нехай $w = y_{j_1} \cdot y_{j_2} \cdot \ldots \cdot y_{j_k}$ – програма, яка є послідовністю операторів присвоєння. Будемо казати, що $w \in P$, якщо P допускає послідовність обчислень w. Також будемо називати w словом $\boldsymbol{P}$. Власне програма P асоціюється тоді з множиною належних їй слів. Інакше, множина слів – це множина усіх можливих шляхів обчислень P.

Введемо наступні позначення. Через $g(w),\ g \in A[X]$, позначимо поліном $g(F_k(F_{k-1}(...(F_1(X))...))$, а через $g(P)$ –множину поліномів виду $g(w)$, $w \in P$. Ідеал, породжений множиною $g(P)$, позначимо через $(g(P))$.

Покажемо, що для кожної програми P існує й може бути ефективно побудована скінченна підмножина слів $W = W(g,P)$ така, що $(g(P)) = (g(W))$. Оскільки g – інваріант програми P тоді й тільки тоді, коли $g(w) = 0$ для кожного $w \in P$, перевірка інваріантності g зводиться до перевірки рівностей $g(w) = 0,\ \ w \in W$.

1. (базис індукції) $P = y_k$. Тоді $W(g,P) = \{y_k\}$.

2. (крок індукції)

а). $P = P_1 \vee P_2$. $W(g, P_1 \vee P_2) = W(g,P_1) \cup W(g,P_2)$

б) $P = P_1 * P_2$. Нехай $W(g,P_2) = \{v_1,...,v_l\}$. Оскільки з $w \in P$ випливає $w = w_1 * w_2$, де $w_i \in P_i$, маємо $g(w) = g(w_1 * w_2) = g(w_2(w_1))$, звідки

$$W(g, P_1 * P_2) = W(g(v_1), P_1) \cup ... \cup W(g(v_l), P_1) \qquad (6.1\ 7)$$

в) $P=\{P_1\}$. Тоді $P=E\cup P_1\cup P_1^2\cup\ldots$, де E – позначення тотожного оператора. Розглянемо послідовність ідеалів:

$$(g)\subseteq(g,g(P_1))\subseteq\ldots\subseteq(g,g(P_1)),\ldots,g(P_1^m))\subseteq\ldots$$

Оскільки кільце $A[X]$ є нетеровим, ця послідовність стабілізується на деякому номері m_0. Покажемо, що m_0 – найменше натуральне число таке, що

$$g(P_1^{m_0+1})\subseteq(g,g(P_1),\ldots,g(P_1^{m_0})) \qquad (6.2\ 8)$$

Насправді, з (6.1) випливає

$$g(P_1^{m_0+2})\subseteq(g(P_1),\ldots,g(P_1^{m_0+1}))\subseteq(g,g(P_1),\ldots,g(P_1^{m_0}))$$

тому з рівності $(g,\ldots,g(P_1^{m_0}))=(g,\ldots,g(P_1^{m_0}),g(P_1^{m_0+1}))$ випливають рівності $(g,\ldots,g(P_1^{m_0}))=(g,\ldots,g(P_1^{m_0+k}))$ для будь-якого натурального k, звідки

$$W(g,P)=W(g,E)\cup W(g,P_1)\cup\ldots\cup W(g,P_1^{m_0}) \qquad (6.3.\ 9)$$

Оскільки проблема приналежності $g\in(q_1,\ldots,q_l)$ алгоритмічно розв'язувана, формули (6.1)-(6.3) описують рекурсивний алгоритм побудови $W(g,P)$. Твердження а) доведене.

Для доведення твердження б) необхідно розглянути поліном $g(X)=a_0X_1^M+\ldots+a_{\varphi(M)}$ ступеня M з невизначеними коефіцієнтами й для нього побудувати множину $W(g,P)$. Оскільки кільце $A[a_0,\ldots,a_{\varphi(M)},X]$ також є нетеровим, цю побудову можна здійснити за допомогою алгоритма пункту а).

Нехай $W(g,P)=\{g_1,\ldots,g_l\}$, $g_k\in A[a_0,\ldots,a_{\varphi(M)},X]$. Система рівностей $g_1\equiv g_2\equiv\ldots\equiv g_l\equiv 0$ визначає систему лінійних рівнянь над A з невідомими $a_0,\ldots,a_{\varphi(M)}$, фундаментальна система розв'язків якої задає шуканий базис $I_P^{(M)}$.

Історично перше доведення наведено у [92], [93]. Алгоритм, викладений у [93], використовує теорію виключення [19]. В [94] запропоновано більш ефективний алгоритм, оснований на використанні базисів Гребнера. Крім того, у цій роботі наведено ефективний на практиці метод невизначених коефіцієнтів зведення проблеми b) до проблеми a). Алгоритми обчислення інваріантів програм та простих циклів наведені також у [95-98].

Приклад 6.4. Розглянемо метод теореми 6.2 на прикладі доведення інваріантності рівності $(x^2-x\cdot y-y^2)^2=1$ програми, що обчислює найменше число Фібоначчі, більше, ніж N.

```
Procedure Fib(N: Integer; var x: Integer);
var y: Integer;
Begin
    x := 1; y := 1;
    While x < N do Begin
      x := x + y;
```

```
      y := x - y
    End {Invariant: (x^2 - x*y - y^2)^2 - 1}
End;
```

Побудуємо словник операторів:

$$y_1 : x := 1; y := 1, \quad y_2 : x := x + y; y := y, \quad y_3 : x := x; y := x - y.$$

Побудуємо регулярний вираз програми: $P = y_1 \cdot \{y_2 \cdot y_3\}$

Послідовне застосування обчислюючих операторів у циклі перетворимо в один оператор $y_4 : x := x + y;\ y := x.$ Регулярний вираз програми при цьому спроститься: $P = y_1 \cdot \{y_4\}$. Інваріант програми представимо у вигляді добутку

$$g(x, y) = (x^2 - xy - y^2 - 1)(x^2 - xy - y^2 + 1).$$

Далі, $P = P_1 \cdot P_2, P_1 = y_1, P_2 = \{y_4\}$. Будуємо $W(P_2, g)$. $P_2 = e \vee y_4 \vee y_4^2 \vee \ldots.$ Знаходимо номер стабілізації ланцюжка ідеалів

$$(g) \subseteq (g, g(y_4)) \subseteq (g, g(y_4), g(y_4^2)) \subseteq \ldots.$$

Обчислимо

$$g(y_4) = g(x + y, x) = ((x + y)^2 - (x + y)x - x^2 - 1) \cdot ((x + y)^2 - (x + y)x - x^2 + 1) =$$

$$= (-x^2 + xy + y^2 - 1)(-x^2 + xy + x^2 + 1) = (x^2 - xy - y^2 + 1)(x^2 - x \cdot y - x^2 - 1).$$

Отже, $g(x, y) = g(x + y, x)$. Таким чином, $m_0 = 0$ і $g(\{y_4\}) = (g)$. Ми довели, що $g(x, y)$ – інваріант циклу. Відзначимо, що якщо g – інваріант циклу а y-тіло циклу, то $g(y)$ – також інваріант циклу. Цей факт можна використовувати. Наприклад, якщо ідеал інваріантів циклу I_C породжується одним поліномом $g(X)$, ланцюжок ідеалів обірветься вже на першому кроці. У протилежному випадку отримаємо новий інваріант.

Нехай $g_1(x, y) = x^2 - x \cdot y - y^2 - 1,\ g_2(x, y) = x^2 - x \cdot y - y^2 + 1.$ При перетвореннях циклу $g_1(x, y)$ переходить у $-g_2(x, y)$, а $g_2(x, y)$ – в $-g_1(x, y)$. Таким чином, ідеал інваріантів цього прикладу не є простим. Він розкладається у перетині простих ідеалів, породжених відповідно поліномами g_1 та g_2.

Далі обчислюємо $W(P, g)$. Отримуємо $W(P_2, g) = \{e\}$. Тому $W(P, g) = \{y_1\}$.

Для доведення інваріантності g треба перевірити, що $g(y_1) \equiv 0.$

Підставляючи $x = 1, y = 1$ в $g(x, y)$, переконуємося в цьому: $g(1,1) = 0.$

Приклад 6.5. Обчислення інваріантів методом невизначених коефіцієнтів. Процедура цього прикладу здійснює поворот точки (x, y) на кут $\arctan(3/4)$ доти, поки вона не потрапить в e-околицю свого початкового положення (a, b).

```
Procedure Rotation (a, b: Real; e: Real; var x, y:
Real);
var u, v: Real;
begin
     x := a; y := b;
     Repeat
        u := (4/5)*x - (3/5)*y;
        v := (3/5)*x + (4/5)*y;
        x := u; y := v;
     until Sqr((x-a)) + Sqr(y-b) < e;
end;
```

Сформуємо словник операторів, редукуючи оператор у тілі циклу до виду

$$y_1 : x := \frac{4}{5} \cdot \mathrm{x} - \frac{3}{5} \cdot \mathrm{y};\ \mathrm{y} := \frac{3}{5} \cdot \mathrm{x} + \frac{4}{5} \cdot \mathrm{y}.\quad y_2 : x := a;\ \mathrm{y} := \mathrm{b}.$$

Програму представлено регулярним виразом $P = y_2 \cdot y_1 \cdot \{y_1\}$.

Будемо шукати інваріант у вигляді однорідного поліному 2-го ступеня від змінних x, y, a, b.

$$g(a,b,x,y) = A \cdot x^2 + B \cdot xy + C \cdot y^2 + D \cdot a^2 + E \cdot ab + F \cdot b^2 \qquad (6.4)$$

На практиці можна використовувати метод, намічений в цьому й наступному прикладах. Відзначимо, що обчислення вздовж будь-якого шляху програми, що виконуються для конкретних числових значень вхідних параметрів, приводять до лінійних рівнянь від невизначених коефіцієнтів $A, B, .., F$. Розглянемо обчислення вздовж шляху $w_0 = y_1 \cdot y_2$

Покладемо

$a = 1, b = 0$. Одержимо: $g(w_0(1,0)) = \frac{16}{25}A + \frac{12}{25}B + \frac{9}{25}C + D = 0$

(6.5)

$a = 0, b = 1$. Одержимо: $g(w_0(0,1)) = \frac{9}{25}A - \frac{12}{25}B + \frac{16}{25}C + F = 0$ (6.6)

$a = 1, b = 1$. Одержимо: $g(w_0(1,1)) = \frac{24}{25}A - \frac{7}{25}B - \frac{24}{25}C - E = 0$ (6.7)

Виключаючи з (6.4) D, E, F, отримаємо:

$$g = A(x^2 - \frac{16}{25}a^2 - \frac{9}{25}b^2 + \frac{24}{25}ab) + B(xy - \frac{12}{25}a^2 + \frac{12}{25}b^2 - \frac{7}{25}ab) + C(y^2 - \frac{9}{25}a^2 - \frac{16}{25}b^2 - \frac{24}{25}ab) \qquad (6.8)$$

Рівняння (6.5)-(6.7) лінійно незалежні. Однак, подальші обчислення показують, що рівняння, отримані при інших початкових значеннях a,b уздовж шляху w_0, залежать від них. Таким чином, можливості отримання нових залежностей з використанням шляху w_0 вичерпані.

Розглянемо обчислення уздовж шляху $w_1 = y_1 y_2 y_2$. Покладемо $a = 1, b = 0$. З (6.8) одержимо:

$$g(w_1(1,0)) = \frac{351}{625}A - \frac{132}{625}B + \frac{351}{625}C = 0 \qquad (6.9)$$

Знаходячи з (6.6) коефіцієнт С, отримаємо:

$$g = A(x^2 - a^2 - b^2 + y^2) + B(xy - \frac{8}{13}a^2 + \frac{28}{117}b^2 - \frac{25}{39}ab + \frac{44}{117}y^2) \qquad (6.10)$$

Покладемо $a = 1, b = 1$. З (6.10) отримаємо:

$$g(w_1(1,1)) = -\frac{50}{39}B = 0 \qquad (6.11)$$

Знаходячи з (6.11) коефіцієнт B, отримаємо:

$$g = A(x^2 - a^2 - b^2 + y^2)$$

Нарешті, методом а) теореми 6.2 переконаємося в тому, що $g = x^2 + y^2 - a^2 - b^2$ – інваріант програми.

Основні задачі і алгоритми методу

З доведення п. а) теореми 6.2 випливає, що основною алгоритмічною проблемою у реалізації відповідного алгоритму є проблема включення:

$$g(P_1^{m+1}) \subseteq (g, g(P_1), ..., g(P_1^m))$$

Іншими словами, проблема полягає у розпізнанні приналежності деякого полінома ідеалу, заданому своїм базисом. Ця проблема є класичною в теорії поліноміальних ідеалів, і для її розв'язання як це показано у п. 5.3, краще за все використати представлення ідеалу базисом Гребнера.

Отже, всі ідеали, які будуються в алгоритмі п. а), будемо вважати представленими своїми базисами Гребнера. Цей факт позначається рівністю $I = (f_1,..., f_k)_{Gr}$. Базис Гребнера ідеала I будемо позначати через $Gr(I)$.

Алгоритм теореми а) базується на алгоритмах розв'язання таких проблем:

1. Для $I = (f_1,..., f_k)_{Gr}$, $J = (g_1,..., g_l)_{Gr}$ знайти $K = (f_1,..., f_k, g_1,...g_l)_{Gr}$.

2. Для $I=(f_1,...,f_k)_{Gr}$, оператора $X:=H(X)$ знайти $Gr((f_1(H),...,f_k(H)))$.

3. Для $I=(f_1,...,f_k)_{Gr}$, $J=(g_1,...,g_l)_{Gr}$ розпізнати включення $I\subset J$.

Проблеми 3 є класичною для теорії базисів Гребнера поліноміальних ідеалів, їх розв'язання наведено у п.5.3. Розв'язання задач 1, 2. полягає у застосуванні алгоритма побудови базису Гребнера відповідно до $(f_1,...,f_k,g_1,...g_l)$ і $(f_1(H),...,f_k(H))$.

Розглянемо тепер питання про реалізацію алгоритма п. b) теореми 6.2. Зауважимо, що метод доведення теореми дозволяє шукати інваріанти за шаблоном, тобто інваріанти довільної поліноміальної форми, визначеної поліномом з невизначеними коефіцієнтами. Наприклад, це може бути поліноміальна форма, задана лінійною комбінацією

$$G(X,a_1,...,a_n)=a_1g_1(X)+...+a_ng_n(X) \tag{6.12}$$

Оскільки підстановка обчислюючих операторів у (6.12) здійснюється для кожного поліному $g_i(X)$ окремо, невизначені коефіцієнти a_i «залишаються на своїх місцях». Таким чином, алгоритм п. а) теореми 6.2 може бути застосований до вектора поліномів $(g_1(X),...,g_n(X))$. Базиси Гребнера можна будувати для окремо для кожного з i наборів поліномів $(g_i(X),g_i(H(X),...,g_i(H^m(X))$, а обчислення номера m_0 стабілізації ланцюжків ідеалів треба обчислювати для всього вектора ідеалів $(I_1,...,I_n)$. Для зменшення значення n треба використовувати метод невизначених коефіцієнтів, продемонстрований у прикладі 6.5.

Інваріанти раціонально визначених програм.

Метод теореми 6.2 можна модифікувати таким чином, щоб він був пригодним до аналізу программ, обчислюючі оператори яких є раціональними виразами з $F(X)$ [94]. Обчислюючі оператори програми мають вид $X:=H(X)$, $H(X)=(h_1(X),...,H_n(X)$, $h_i(x)=r_i(X)/s_i(X), i=1,...,n$, причому $h_i(X)$ – нескорочувані дроби. Разом зі змінними набору $X=<x_1,...,x_n>$ розглянемо набір змінних $Z=<z_1,...,z_n>$ і набір поліномів

$$f_1(Z),...,f_k(Z),x_1\cdot s_1(Z)-r_1(Z),...,x_n\cdot s_n(Z)-r_n(Z) \tag{6.13}$$

Побудуємо базис Гребнера набору (6.13), елімінуючи у першу чергу множину змінних набору Z. Отриманий базис Гребнера є розв'язком проблеми 2 переліку основних проблем методу. Аналогічно можна розв'язувати проблему 2 і для обчислюючи операторів, які містять у правих частинах радикали.

Приклад 6.6. У цьому прикладі ми розглянемо обчислення інваріантів програми, що узагальнює програму попереднього прикладу. Саме,

аналізується процедура повороту точки (a,b) на довільний кут α. Цей кут задається вхідним параметром $t = \tan(\frac{\alpha}{2})$.

```
{t = tan(alfa/2),      alfa – кут повороту}
Procedure Rotation (a, b: Real; t: Real; e: Real;
var x, y: Real);
var u, v: Real;
    c, s: Real;
begin
     c := (1-sqr(t))/(1+sqr(t));  {c = cos(alfa)}
    s := (2*t)/(1 + sqr(t)); {s = sin(alfa)}
    x := a; y := b;
     Repeat
        u := c*x - s*y;
        v := s*x + c*y;
        x := u; y := v;
     until Sqr((x-a)) + Sqr(y-b) < e;
end;
```

Сформуємо словник обчислюючих операторів:

$y_1 : c := (1-t^2)/(1+t^2), s := 2t/(1+t^2), x := a;\ \mathrm{y} := \mathrm{b}.$

$y_2 : x := \mathrm{cx} - \mathrm{sy};\ \mathrm{y} := \mathrm{sx} + c\mathrm{y}.$

Програма представлена регулярним виразом $P = y_1 y_2 \{y_2\}$. Як і в попередньому прикладі, будемо шукати інваріант у вигляді

$$g(a,b,x,y) = Ax^2 + Bxy + Cy^2 + Da^2 + Eab + Fb^2$$

Можливі декілька підходів до обчислення інваріантів. По-перше, можна застосувати метод невизначених коефіцієнтів попереднього прикладу. Для цього потрібно надавати різні числові значення змінним a,b,t. По-друге, обчислення, аналогічні обчисленням попереднього прикладу, можна вести над полем $Q(t)$. Нарешті, можна обчислити інваріанти після застосування оператора y_1, використовуючи метод п.6 виключення змінних для раціонально-визначених обчислюючих операторів. Опишемо більш детально цей підхід.

Для оператора $y_1 : c := (1-t^2)/(1+t^2), s := 2t/(1+t^2), x := a;\ \mathrm{y} := \mathrm{b}$ складемо систему поліномів (6.14). Отримаємо: $(c+1)t^2 + (c-1), st^2 - 2t + s, x-a, y-b$. Елімінуємо змінну t. Отримаємо: $c^2 + s^2 - 1, x-a, y-b$. Подальші обчислення виконуємо над фактор-полем $Q(a,b,c)[s]/(c^2 + s^2 - 1)$.

Розглянемо обчислення уздовж шляху $w_0 = y_1 y_2$. . У праві частини оператора y_2 підставимо a замість x, b замість y. Всі результати представимо системою рівностей:

$$\begin{cases} g = Ax^2 + Bxy + Cy^2 + Da^2 + Eab + Fb^2 \\ x = ca - sb \\ y = sa + cb \\ c^2 + s^2 = 1 \end{cases} \tag{6.14}$$

Підставимо значення x, y в першу рівність системи (6.14) і приведемо поліном $g(a,b,c,s)$ до стандартного виду. Приведемо поліном g за модулем $c^2 + s^2 - 1$, замінюючи c^2 на $1 - s^2$. Приведемо отриманий поліном $g(a,b,c,s)$ до стандартного виду й дорівняємо нулю його коефіцієнти. Отримаємо:

$$B = 0, A - C = 0, C + D = 0, F + A = 0, E = 0.$$

Виключивши з (6.11) коефіцієнти B, C, D, E, F , отримаємо:

$$g(x, y, a, b) = A(x^2 + y^2 - a^2 - b^2).$$

Залишилося довести, що $g = x^2 + y^2 - a^2 - b^2 -$ інваріант. Зведемо в систему всі необхідні для обчислень рівності

$$\begin{cases} u^2 + v^2 = a^2 + b^2 \\ u = cx - sy \\ v = sx + cy \\ c^2 + s^2 = 1 \end{cases} \tag{6.15}$$

Підставимо в першу рівність (6.15) значення u, v й приведемо отриману рівність за модулем $c^2 + s^2 - 1$. Отримаємо: $x^2 + y^2 = a^2 + b^2$.

Відзначимо, що проблема побудови базису I_P залишається відкритою.

6.3. Задача обчислення інваріантів лінійних ділянок програм

У найбільш простих випадках задачу обчислення інваріантів треба розв'язувати для лінійних (нерозгалужених) програм. або для лінійної послідовності операторів присвоєння, яку називають лінійною ділянкою програми. У цьому пункті наведено розв'язання цієї задачі. Зауважимо, що розв'язання цієї задачі дає змогу обчислювати деякі інваріанти у більш складних випадках, наприклад, деяких інваріантів циклу.

Означення 6.2. Нехай $X = (x_1, x_2, ..., x_n)$ – вектор змінних**.** Набір поліномів $f_1(X), ..., f_m(X) \in F[X]$ називається алгебраїчно залежним, якщо існує поліном $A \in F[u_1, ..., u_m]$ такий, що $A(f_1(X), ..., f_m(X)) \equiv 0$.

Обгрунтування методу полягає у наступному. Лінійну ділянку програми можна замінити одним векторним оператором присвоєння

$$x_1 := f_1(x_1, ..., x_n);$$
$$x_2 := f_2(x_1, ..., x_n);$$
$$...$$
$$x_2 := f_n(x_1, ..., x_n).$$

Тоді, якщо поліноми $f_1(X), ..., f_m(X)$ алгебраїчно залежні, тобто існує такий поліном $A \in F[u_1, ..., u_m]$, що $A(f_1(X), ..., f_m(X)) \equiv 0$, очевидно, що $A(x_1, x_2, ..., x_n)$ – інваріант цієї ділянки.

Теорема 6.3. Нехай $X = (x_1, ..., x_n)$, $f_1(X), ..., f_m(X) \in F[X]$ і $m > n$. Тоді набір $f_1(X), ..., f_m(X)$ є алгебраїчно залежним.

Доведення наведено у [109].

Наслідок. Якщо для деякого набору змінних $X' \subset X$ цикл має вид

While U **do** X := (f₁(X′), …, fₙ(X′)). (6.16)

Тоді цей цикл має нетривіальний поліноміальний інваріант.

Теорема 6.4. Нехай набір поліномів $f_1(X), ..., f_m(X) \in F[X]$ є алгебраїчно залежним. Тоді ідеал $I = \{A \in F[U] \mid A(f_1(X), ..., f_m(X)) \equiv 0\}$ має базис Гребнера, який визначено співвідношенням $(u_1 - f_1(X), ..., u_m - f_m(X))_{Gr} \cap F[U]$.

Наслідок. Алгоритм обчислення інваріантів циклу 6.16 полягає в обчисленні базису Гребнера елімінацією змінних множини X.

Приклад 6.7. Цикл з нетривіальним інваріантом.

While U **do** (x,y,z) := (x-y,x*y,x+y).

Складемо набір поліномів $(u_1 - x + y, u_2 - xy, u_3 - x - y)$. Обчислимо базис Гребнера цього набору, елімінуючи змінні x, y. Отримаємо поліном $4u_2 + (u_1 + u_3)(u_1 - u_3)$. Здійснимо заміну змінних $u_1 = x, u_2 = y, u_3 = z$. Отримаємо інваріант циклу

$$A(x, y, z) = 4y + (x + z)(x - z).$$

6.4. Метод L-інваріантів генерації інваріантів лінійних циклів

6.4.1. L-інваріанти простих лінійних циклів

Один з простих, але розповсюджених частинних випадків полягає у тому, що цикл імперативної програми є лінійним.

Означення 6.3. Нехай $X = \{x_1,...,x_n\}$, $b = \{b_1,...,b_n\}$ – два набори змінних. Лінійним циклом називається фрагмент імперативної програми виду

```
X := b;
While Q(X, b) do X := A*X
```

Метод обчислення інваріантів цього пункту будує так звані L-інваріанти лінійних циклів. Основні результати теорії L-інваріантів циклів викладено у [100].

Означення 6.4. Нехай W – n-мірний векторний простір на полі раціональних чисел Q і $\bar{Q}$ – алгебраїчне замикання Q. Позначимо через $X = (x_1,....x_n)$ n-мірний вектор змінних. Раціональна функція $p(X) \in \bar{Q}(X)$ називається L-інваріантом лінійного оператора $A: W \to W$, якщо для будь-якого $b \in W$ має місце співвідношення

$$p(A \cdot b) = p(b).$$

Приклад 6.8. **(**лінійний оператор з характеристичним поліномом $x^3 - 2$). Розглянемо лінійний оператор з матрицею

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 2 & 0 & 0 \end{pmatrix}, X = (x, y, z) .$$

Покажемо, що раціональний вираз

$$p(x, y, z) = \frac{(\lambda_1^2 x + \lambda_1 y + z)(\lambda_3^2 x + \lambda_3 y + z)}{(\lambda_2^2 x + \lambda_2 y + z)^2} \tag{6.17}$$

де $\lambda_1 = \sqrt[3]{2}, \lambda_2 = \sqrt[3]{2}\varepsilon, \lambda_3 = \sqrt[3]{2}\varepsilon^2$, а $\varepsilon = \cos(\frac{2\pi}{3}) + i\sin(\frac{2\pi}{3})$ – первісний корінь ступеня 3 з 1 – L- інваріант цього оператора.

$$p(A \cdot (x, y, z)^T) = \frac{(\lambda_1^2 y + \lambda_1 z + 2x)(\lambda_3^2 y + \lambda_3 z + 2x)}{(\lambda_2^2 y + \lambda_2 z + 2x)^2} =$$

$$= \frac{\lambda_1(\lambda_1 y + z + \lambda_1^2 x)\lambda_3(\lambda_3 y + z + \lambda_3^2 x)}{\lambda_2^2(\lambda_2 y + z + \lambda_2^2 x)^2} =$$

$$= \frac{\lambda_1\lambda_3}{\lambda_2^2} \cdot \frac{(\lambda_1^2 x + \lambda_1 y + z)(\lambda_3^2 x + \lambda_3 y + z)}{(\lambda_2^2 x + \lambda_2 y + z)^2} =$$

$$= \frac{(\lambda_1^2 x + \lambda_1 y + z)(\lambda_3^2 x + \lambda_3 y + z)}{(\lambda_2^2 x + \lambda_2 y + z)^2}.$$

Зауваження. Надалі умову $Q(X,b)$ ми будемо ігнорувати, вважаючи лінійний цикл нескінченним, а його виконання недетермінованим. Таким чином, ми розглядаємо цикли виду

```
X := b;
While True|False do X := A*X
```

Теорема 6.5. Якщо $p(X) = \dfrac{r(X)}{q(X)}$ – L-інваріант лінійного оператора A, поліном $r(X)q(b) - q(X)r(b)$ – інваріант лінійного циклу над полем $\bar{Q}$. Такі інваріанти циклів ми будемо також називати L-інваріантами (лінійних циклів).

Приклад 6.9. (лінійний цикл з оператором прикладу 6.8).

Лінійний цикл, що відповідає операторові A, має вигляд

```
(x, y, z) := (a, b, c);
While True|False do (x, y, z) := (y, z, 2*x)
```

L- інваріант цього циклу визначений формулою (6.17):

$$P(x,y,z,a,b,c) = (\lambda_1^2 x + \lambda_1 y + z)(\lambda_3^2 x + \lambda_3 y + z)(\lambda_2^2 a + \lambda_2 b + c)^2 - \\ - (\lambda_2^2 x + \lambda_2 y + z)^2(\lambda_1^2 a + \lambda_1 b + c)(\lambda_3^2 a + \lambda_3 b + c) \qquad (6.18)$$

Відзначимо, що L-інваріант циклу $P(x,y,z,a,b,c)$ визначений над полем $\bar{Q}(\lambda_1,\lambda_2,\lambda_3)$. Однак йому відповідає набір L-інваріантів з коефіцієнтами з поля Q, які можна побудувати, звівши (6.15) до канонічного виду – поліному від $\lambda_1,\lambda_2,\lambda_3$, а потім – до поліному від λ_2, використовуючи співвідношення $\lambda_1\lambda_3 = \lambda_2^2$. Продемонструємо техніку обчислення L-інваріантів над полем Q.

Введемо позначення

$$r(x,y,z) = (\lambda_1^2 x + \lambda_1 y + z)(\lambda_3^2 x + \lambda_3 y + z),\ q(x,y,z) = (\lambda_2^2 x + \lambda_2 y + z)^2$$

Поліноми r, q визначені над полем $Q(\lambda_2)$. Обчислимо $r(x,y,z), q(x,y,z)$. Отримаємо:

$$r(x,y,z) = r_0(x,y,z) + r_1(x,y,z)\lambda_2 + r_2(x,y,z)\lambda_2^2,$$
$$q(x,y,z) = q_0(x,y,z) + q_1(x,y,z)\lambda_2 + q_2(x,y,z)\lambda_2^2$$

де

$$r_0(x,y,z) = z^2 - 2xy,\ r_1(x,y,z) = 2x^2 - yz,\ r_2(x,y,z) = y^2 - xz.$$

$$q_0(x,y,z) = z^2 + 4xy,\ q_1(x,y,z) = x^2 + yz,\ q_2(x,y,z) = y^2 + 2xz.$$

Дріб $\dfrac{r(x,y,z)}{q(x,y,z)}$ представимий у вигляді поліному від λ_2 з коефіцієнтами з $Q(x,y,z)$.

$$\frac{r(x,y,z)}{q(x,y,z)} = \frac{r_0(x,y,z) + r_1(x,y,z)\lambda_2 + r_2(x,y,z)\lambda_2^2}{q_0(x,y,z) + q_1(x,y,z)\lambda_2 + q_2(x,y,z)\lambda} = U + V\lambda_2 + W\lambda_2^2, \qquad (6.19)$$

$$U, V, W \in Q(x,y,z).$$

$U(x,y,z), V(x,y,z), W(x,y,z)$ можна обчислити методом невизначених коефіцієнтів, використовуючи рівність (6.16). Тоді $U(x,y,z), V(x,y,z), W(x,y,z)$ є L-інваріантами оператора A. Справді,

$$p(x,y,z) - p(a,b,c) =$$
$$(U(x,y,z) - U(a,b,c)) + (V(x,y,z) - V(a,b,c))\lambda_2 + (W(x,y,z) - W(a,b,c))\lambda_2^2$$

Оскільки $p(b,c,2a) + p(a,b,c) = 0$, маємо:

$$(U(b,c,2a) - U(a,b,c)) + (V(b,c,2a) - V(a,b,c))\lambda_2 + (W(b,c,2a) - W(a,b,c))\lambda_2^2 = 0$$

Через лінійну незалежність векторів $1, \lambda_2, \lambda_2^2$ над Q отримаємо:

$$U(b,c,2a) - U(a,b,c) = 0,\ V(b,c,2a) - V(a,b,c) = 0,\ W(b,c,2a) - W(a,b,c) = 0$$

Тобто $U(x,y,z),\ V(x,y,z),\ W(x,y,z)$ – L-інваріанти циклу над Q.

Метод побудови L- інваріантів, який ми розглянемо, полягає в обчисленні й аналізі власних значень і власних векторів лінійних операторів.

Теорема 6.6. Нехай $\lambda_1,...,\lambda_m$ – власні значення лінійного оператора A й $s_1,...,s_m$ – відповідні їм власні вектори сполученого оператора A^*. Припустимо, що існують такі цілі числа $k_1,...,k_m$, що

$$\lambda_1^{k_1} \cdot ... \cdot \lambda_m^{k_m} = 1 . \tag{6.20}$$

Тоді

$$p(X) = (s_1, X)^{k_1} \cdot ... \cdot (s_m, X)^{k_m} \tag{6.21}$$

– L - інваріант лінійного оператора A .

Доведення. Нехай $\lambda_1^{k_1} \cdot ... \cdot \lambda_m^{k_m} = 1$. Тоді

$$(s_1, AX)^{k_1} \cdot ... \cdot (s_m, AX)^{k_m} = (s_1 A, X)^{k_1} \cdot ... \cdot (s_m A, X)^{k_m} = (A^* s_1, X)^{k_1} \cdot ... \cdot (A^* s_m, X)^{k_m} =$$
$$= (\lambda_1 s_1, X)^{k_1} \cdot ... \cdot (\lambda_m s_m, X)^{k_m} = \lambda_1^{k_1} \cdot ... \cdot \lambda_m^{k_m} (s_1, X)^{k_1} \cdot ... \cdot (s_m, X)^{k_m} = (s_1, X)^{k_1} \cdot ... \cdot (s_m, X)^{k_m}.$$

Співвідношення (6.20) будемо називати мультиплікативним співвідношенням (між коріннями характеристичного полінома), що визначає L-інваріант (6.21).

Приклад 6.10. (продовження прикладу 6.9).

Розглянемо метод теореми 6.6 на прикладі 6.8. Обчислимо власні числа оператора A.

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 2 & 0 & 0 \end{pmatrix}, \; h(\lambda) = |A - \lambda E| = \begin{vmatrix} -\lambda & 1 & 0 \\ 0 & -\lambda & 1 \\ 2 & 0 & -\lambda \end{vmatrix} = -\lambda^3 + 2 .$$

Характеристичний поліном має вигляд $h(x) = x^3 - 2$. Його корені –

$$\lambda_1 = \sqrt[3]{2}, \; \lambda_2 = \sqrt[3]{2}\varepsilon, \; \lambda_3 = \sqrt[3]{2}\varepsilon^2, \; \varepsilon = \cos(\frac{2\pi}{3}) + i\sin(\frac{2\pi}{3}),$$

де ε – первісний корінь ступеня 3 з 1.

Обчислимо далі власні вектори матриці $A^* = \begin{pmatrix} 0 & 0 & 2 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$. Розв'яжемо для цього систему однорідних лінійних рівнянь $(A^* - \lambda E) \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} = 0$, причому обчислення будемо здійснювати у полі $Q(\lambda)$ за модулем $\lambda^3 - 2$. Отримаємо

систему лінійних рівнянь $\begin{cases} \lambda x - 2z = 0 \\ x - \lambda y = 0 \\ y - \lambda z = 0 \end{cases}$, ранг якої дорівнює 2. Фундаментальний розв'язок цієї системи – вектор $s = (\lambda^2, \lambda, 1)$. Тому власними векторами оператора A^* є вектори

$$s_1 = (\lambda_1^2, \lambda_1, 1),\ s_2 = (\lambda_2^2, \lambda_2, 1),\ s_3 = (\lambda_3^2, \lambda_3, 1).$$

Легко встановити, що $\frac{\lambda_1 \lambda_3}{\lambda_2^2} = 1$. Тому оператор A має L - інваріант (6.17).

6.4.2. Класи лінійних операторів з нетривіальними інваріантами

Наслідок 1. Якщо вільний член мінімального характеристичного полінома $h(x)$ лінійного оператора A дорівнює ± 1, лінійний оператор має L -інваріант.

Доведення. Нехай c – вільний член полінома $h(x)$. Тоді $c = \lambda_1 \cdot ... \cdot \lambda_m = \pm 1$. Тому або $(s_1, X) \cdot ... \cdot (s_m, X)$, або $(((s_1, X) \cdot ... \cdot (s_m, X))^2$ $-L$-інваріант оператора A. Відзначимо, що коефіцієнти цього поліному належать Q, оскільки вони симетричні відносно перестановок коренів $\lambda_1, ..., \lambda_m$.

Оскільки вільний член $h(x)$ дорівнює визначнику A, наявність L-інваріанту у цьому випадку можна обчислити за час $O(n^3)$.

Приклад 6.11. Цикл повороту точки площини (a,b) на кут $\arctan(\frac{4}{3})$.

```
  (x,y) := (a,b);
  While True|False do (x, y):= (4/5*x-3/5*y,
3/5*x+4/5*y)
```

Обчислимо власні значення й власні вектори оператора A:

$$A = \begin{pmatrix} 4/5 & -3/5 \\ 3/5 & 4/5 \end{pmatrix}.\ h(\lambda) = |A - \lambda E| = \begin{vmatrix} 4/5 - \lambda & -3/5 \\ 3/5 & 4/5 - \lambda \end{vmatrix} = \lambda^2 - \frac{8}{5}\lambda + 1.$$

$$\lambda_1 = \frac{4}{5} - i\frac{3}{5}, \lambda_2 = \frac{4}{5} + i\frac{3}{5}.\ s_1 = (i, 1),\ s_2 = (-i, 1).$$

Оскільки $\lambda_1 * \lambda_2 = 1$, L-інваріант оператора A має вигляд

$$p(x, y) = (ix + y)(-ix + y) = x^2 + y^2.$$

Інваріант циклу має вигляд $x^2 + y^2 - a^2 - b^2$.

Приклад 6.12. Цикл обчислення послідовності Фібоначчі, починаючи з пари (a,b).

```
(x,y) := (a,b);
While True|False do (x, y):= (x + y, x)
```

Обчислимо власні значення й власні вектори оператора A:

$$A=\begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}.\ h(\lambda)=|A-\lambda E|=\begin{vmatrix}1-\lambda & 1\\ 1 & -\lambda\end{vmatrix}=\lambda^2-\lambda-1.$$

$$\lambda_1=\frac{1}{2}-\frac{1}{2}\sqrt{5},\ \lambda_2=\frac{1}{2}+\frac{1}{2}\sqrt{5}.$$

$$s_1=(\lambda_1,1)=(\frac{1}{2}-\frac{1}{2}\sqrt{5},1),\ s_2=(\lambda_2,1)=(\frac{1}{2}+\frac{1}{2}\sqrt{5},1).$$

Оскільки $\lambda_1\lambda_2=-1$, L-інваріант оператора A має вигляд

$$p(x,y)=((\lambda_1 x+y)(\lambda_2 x+y))^2=(x^2-xy-y^2)^2.$$

Інваріантне співвідношення циклу має вигляд $(x^2-xy-y^2)^2=(a^2-ab-b^2)^2$.

Наслідок 2. Якщо характеристичний (мінімальний) поліном $h(X)$ лінійного оператора A має вигляд x^m-a, лінійний оператор має L-інваріант.

Доведення. Корінь λ_i характеристичного полінома x^m-a визначені формулою $\lambda_i=\sqrt[m]{a}\varepsilon^i$, де $\varepsilon-$первісний корінь ступеня m з 1. Легко бачити, що якщо $k_1,...,k_m-$ цілі числа такі, що $k_1+...+k_m=0$, то $\lambda_1^{k_1}\cdot...\cdot\lambda_m^{k_m}-$ деякий ступінь ε. Справді,

$$\lambda_1^{k_1}\cdot...\cdot\lambda_m^{k_m}=(\sqrt[m]{a})^{\sum k_i}\varepsilon^{k_1}\varepsilon^{2k_2}\cdot...\cdot\varepsilon^{mk_m}=(\sqrt[m]{a})^0\varepsilon^K=\varepsilon^K,$$

де $K=\sum ik_i$. Тому добуток $\lambda_1^{k_1}\cdot...\cdot\lambda_m^{k_m}$ у підходящому ступені дорівнює 1. Отже, L-інваріанти оператора A існують і обчислюється аналогічно тому, як це зроблено у прикладі 6.9. Зокрема, L-інваріантами оператора A є раціональні вирази

$$P_i(X)=(\frac{(s_i,X)}{(s_1,X)})^{K_i},i=2,...,m,$$

де s_i – власні вектори A^*, а K_i – найменші натуральні числа такі, що iK_i кратно $m: K_i = im\, div(\gcd(i,m))$.

Теорема 6.7. Нехай $h(x)$ – поліном змінної x з раціональними коефіцієнтами й $\Lambda = (\lambda_1,...,\lambda_m)$ – всіх його коренів із алгебраїчного замикання $\bar{Q}$ поля Q. Розглянемо множину $G(h) = \{x_1^{k_1} \cdot ... \cdot x_m^{k_m} : \lambda_1^{k_1} \cdot ... \cdot \lambda_m^{k_m} = 1\}$ – множина мономів з поля раціональних виразів $Q(X)$ (можливо, з негативними ступенями), які при підстановці λ_i замість x_i отримують значення 1. Тоді $G(h)$ – мультиплікативна абелева група зі скінченним числом твірних.
Доведення з [100] ми опускаємо.

Приклад 6.13. (Продовження прикладу 6.10).

Легко бачити, що для полінома $h(x) = x^3 - 2$ мають місце наступні мультиплікативні співвідношення між його коренями:

$$\lambda_1^2 = \lambda_2\lambda_3,\ \lambda_1\lambda_2 = \lambda_3^2,\ \lambda_1\lambda_3 = \lambda_2^2,\ \lambda_2^3 = \lambda_3^3.$$

Цим співвідношенням відповідають біноми

$$x_1^2 - x_2x_3,\ x_1x_2 - x_3^2,\ x_1x_3 - x_2^2,\ x_2^3 - x_3^3,$$

які утворюють базис Гребнера ідеалу $I(G_B) = I(G(h))$.

Наслідок 1. Нехай $h(x)$ – поліном від змінної x з раціональними коефіцієнтами й $\Lambda = (\lambda_1,...,\lambda_m)$ – множина усіх його коренів із алгебраїчного замикання $\bar{Q}$ поля Q. Розглянемо множину $G_Q(h) = \{x_1^{k_1} \cdot ... \cdot x_m^{k_m} : \lambda_1^{k_1} \cdot ... \cdot \lambda_m^{k_m} \in Q\}$ – множина мономів з поля раціональних виразів $Q(X)$ (можливо, з негативними ступенями), які при підстановці λ_i замість x_i отримують раціональні значення. Тоді $G_Q(h)$ – мультиплікативна абелева група зі скінченним числом твірних.

Наслідок 2. Множина всіх L-інваріантів оператора A утворює поле раціональних виразів.

Проблему опису усіх L-інваріантів лінійного оператора можна тепер уточнити як проблему побудови скінченної множини твірних групи $G(h)$.

Теорема 6.8. Нехай $p(x)$ – приведений незвідний поліном над полем Q, і $\{\lambda_1, \lambda_2, ...\lambda_m\}$ – множина його коренів над полем $\bar{Q}$. Якщо існує нетривіальне мультиплікативне співвідношення $\lambda_1^{k_1} \cdot ... \cdot \lambda_m^{k_m} = 1$ із цілими показниками $k_1,...,k_m$, то вільний член a_m полінома $f(x)$ дорівнює або ± 1, або $\sum_{i=1}^{m} k_i = 0$.

Доведення з [100] ми опускаємо.

Означення 6.5. L-інваріанти оператора A, що відповідають мультиплікативним співвідношенням між коренями характеристичного полінома виду $\lambda_1 \cdot ... \cdot \lambda_m = \pm 1$, будемо називати цілими. L-інваріанти оператора A, що відповідають мультиплікативним співвідношенням $\lambda_1^{k_1} \cdot ... \cdot \lambda_m^{k_m} = 1, \sum k_i = 0$, будемо називати раціональними.

Теорема 6.9. Якщо характеристичний поліном оператора A має вигляд $h(x^k), k > 1$, оператор A має раціональні L-інваріанти.

Доведення. Нехай $\mu_1, ..., \mu_l$ – корені поліному $h(y)$. Тоді $h(x^k) = (x^k - \mu_1)...(x^k - \mu_l)$. Кожний зі співмножників виду $x^k - \mu_i$ визначає раціональні L-інваріанти, які обчислюються аналогічно тому, як це зроблено в наслідку 2 теореми 6.6.

Зауваження. Задачу дослідження L-інваріантів лінійних операторів роботі ще не завершено. Однак лінійні оператори з цілими та раціональними L-інваріантами є гарним джерелом програмних циклів з нетривіальними поліноміальними інваріантами.

6.5. Метод доведення інваріантності лінійних нерівностей лінійних циклів

6.5.1. Інварінтні лінійні нерівності. Постановка задачі.

У попередніх підрозділах основна увага приділялася задачі побудови поліноміальних інваріантів типу рівностей. Множина інваріантів типу рівностей утворює поліноміальний ідеал, скінченний базис якого потрібно побудувати. Тоді доведення інваріантності рівності є стендартною задачею доведення належності полінома ідеалу за допомогою представлення полінома у базисі ідеалу.

Задача доведення інваріантності нерівностей менш вивчена. Основною особливістю є нескінченність базису метаідеалу [105, 108] поліноміальних нерівностей. Ітеративні методи розв'язку задачі побудови деякого класу лінійних інваріантних нерівностей розглядалася у [102-106]. У [107] надано метод доведення інваріантності системи лінійних нерівностей для класу лінійних ітеративних циклів з дійсними власними числами лінійного оператора в тілі циклу. У цьому підрозділі цей метод розширено на клас лінійних ітеративних циклів з довільними лінійними операторами у тілі циклу. Метод використовує алгоритми комп'ютерної алгебри, викладені у попередніх главах.

Нехай K^n – n-мірний векторний простір над лінійно-впорядкованим конструктивним полем K і $\bar{K}$ – алгебраїчне замикання K.

Означення 6.6. Лінійною напівалгебраїчною множиною $M(x_1,...,x_n)$ називається область K^n, певна бесквантерной формулою в сигнатурі логічних зв'язувань $<\vee,\&,\neg>$ із лінійними нерівностями від змінних $x_1,...,x_n$ у якості атомів. Якщо область M задається формулою $F(X)$, тобто $M=\{X : F(X)\}$, будемо позначати її через $M(F(X))$.

Означення 6.7. Нехай $X=(x_1,...,x_n)$, $\bar{b}=(b_1,...,b_n)$ – два вектори змінних. Лінійним циклом з предумовою називається фрагмент імперативної програми виду

```
X := b;  // S(b̄) - передумова
While U(X, b) do X := A*X
```
(6.22)

де $S(\bar{b})$, $U(X,\bar{b})$ – безквантерні формули прикладної логіки лінійних напівалгебраїчних множин, A – матриця лінійного оператора $K^n \to K^n$.

Нетермінованим циклом, асоційованим з циклом (6.22), називається цикл виду

```
X := b;  // S(b̄) - передумова
While True|false do X := A*X
```
(6.23)

кількість повторень якого недетермінована.

Зауваження 1. Оператори X := b, X := A*X інтерпретуються як одночасні присвоєння змінним лівих частин значень правих частин.

Означення 6.8. Послідовність $\{\bar{b}^{(m)}\}$, визначена рекурсивними співвідношеннями

$$\bar{b}^{(0)}=\bar{b},\ \bar{b}^{(m+1)}=A\bar{b}^{(m)},m=0,1,... \tag{6.24}$$

називається орбітою лінійного оператора A з початковою точкою $\bar{b}$ й позначається через $Orbit(A,\bar{b})$.

Означення 6.9. Лінійний цикл (6.19) називається, таким, що завершуються, якщо для будь-якого $\bar{b}\in M(S(X))$ послідовність (6.24) при деякому натуральному $m^*=m^*(\bar{b})$ задовольняє співвідношенню $\neg U(\bar{b}^{(m^*)},\bar{b})$.

Таким чином, якщо цикл завершується, для кожного $\bar{b}\in M(S(X))$ існує найменше натуральне число $m^*(\bar{b})$, на якому цикл (6.22) завершується.

Означення 6.10. Нехай $\bar{a},\bar{c}\in K^n$. Лінійна нерівність

$$L(\bar{a},\bar{c},X,\bar{b})\overset{df}{=}(\bar{a},X)\le(\bar{c},\bar{b}) \tag{6.25}$$

називається умовним інваріантом лінійного циклу (6.19) з передумовою $S(\bar{b})$ (або скорочено – інваріантом), якщо для кожного $\bar{b} \in M(S(X))$ $Orbit(A,\bar{b})$ (6.21) задовольняє співвідношенням:

$$S(\bar{b}) \to L(\bar{a},\bar{c},\bar{b},\bar{b}), \quad U(\bar{b}^{(m-1)},\bar{b}) \to L(\bar{a},\bar{c},\bar{b}^{(m)},\bar{b}),\ m=1,2,...,m^*(\bar{b}).$$

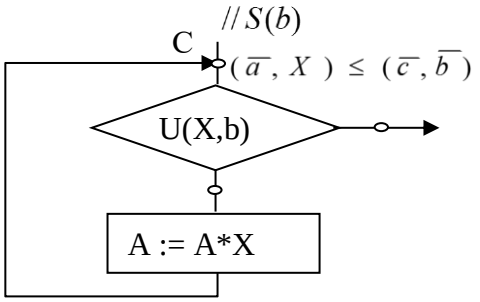


Рис.6.3. Лінійний цикл із контрольною точкою C,
у якій визначений умовний інваріант.

Зауваження 2. Якщо цикл (6.22) не завершується (розходиться) у деякій точці $\bar{b}$, $m^*(\bar{b})$ слід вважати рівним нескінченності: $m^*(\bar{b}) = +\infty$.

Приклад 6.14.

$S(x,y) = (0 \le x \le 1)\,\&\,(0 \le y \le 1)$,
$U(x,y,b_1,b_2) = \neg(|x+b_1| \le \varepsilon)\,\&\,(|y+b_2| \le \varepsilon)$,
$A = \begin{bmatrix} 3/5 & 4/5 \\ -4/5 & 3/5 \end{bmatrix}$.
$L = x + y \le 2b_1 + 2b_2 \,//\, \bar{a} = (1,1),\ \bar{c} = (2,2)$.

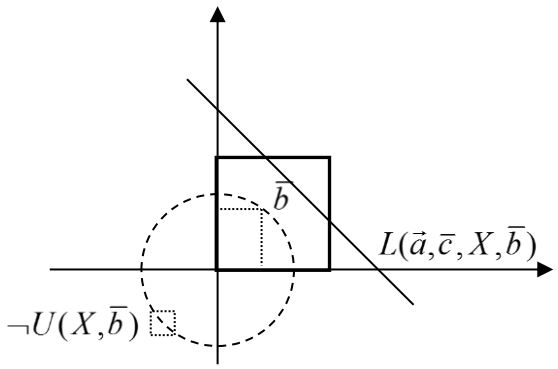


Рис.6.4. Геометрична ілюстрація лінійно-певного циклу.

У цьому прикладі лінійний оператор A є оператором повороту на кут $\alpha = arctg(4/3)$. Початкова точка $\bar{b}$ належить одиничному квадрату. Орбіта лінійного оператора A – послідовність, кожна точка якої лежить на окружності $x^2 + y^2 = b_1^2 + b_2^2$. Умова повторення циклу – «точка (x,y) лежить поза квадратиком зі стороною 2ε й центром у точці $(-b_1,-b_2)$». Тому цикл завершиться, коли точка орбіти потрапить в середину цього квадратика, тобто точка зробить поворот на кут $\pi + 2k\pi$ з точністю ε. Оскільки кут α непорівняний з π, орбіта оператора A – усюди щільна множина на окружності $x^2 + y^2 = b_1^2 + b_2^2$, отже, цикл завершиться. У прикладі 6.16 основний алгоритм застосовується для доведення інваріантності циклу $L = x + y \le 2b_1 + 2b_2$.

Нижче доведено, що проблеми завершення циклу (6.22) і доведення інваріантності лінійної нерівності алгоритмічно розв'язувані для будь-якого лінійного операторра A.

6.5.2. Метод доведення інваріантності лінійної нерівності

Жордановою кліткою називають $m \times m$ матрицю виду

$$J(\lambda) = \begin{bmatrix} \lambda & 1 & \dots & 0 \\ 0 & \lambda & \dots & 0 \\ \cdot & \cdot & \cdot & \cdot \\ 0 & \dots & \lambda & 1 \\ 0 & \dots & 0 & \lambda \end{bmatrix} \tag{6.26}$$

Матриця оператора A в жордановій формі має вигляд

$$A = \begin{bmatrix} J_1(\lambda_1) & 0 & \dots & 0 \\ 0 & J_2(\lambda_2) & \dots & 0 \\ 0 & \dots & J_{k-1}(\lambda_{k-1)} & 0 \\ 0 & \dots & 0 & J_k(\lambda_k) \end{bmatrix},$$

Тобто складена з декількох жорданових кліток різних розмірів, розташованих на головній діагоналі, причому кожна жорданова клітка має своє власне значення λ_j оператора A. Зокрема, якщо лінійний оператор A в тілі циклу (6.22) діагоналізовний, у базисі власних векторів оператора A його матриця має вигляд

$$A = \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \dots & \lambda_n \end{bmatrix}$$

Нехай $f(x)$ – мінімальний характеристичний поліном оператора A, $\Lambda = \{\lambda_1,\dots,\lambda_n\}$ – множина його коренів (спектр A) і $\bar{e}_1,\dots,\bar{e}_n$ – множина власних векторів A. Нехай, далі, $\lambda_1,\dots,\lambda_{2k}$ – множина комплексних власних чисел A, а $\lambda_{2k+1},\dots,\lambda_n$ – множина дійсних власних чисел A, причому $\lambda_1 = \overline{\lambda}_2,\dots,\lambda_{2k-1} = \overline{\lambda}_{2k}$. Визначимо вектори

$$\bar{l}_1 = \frac{1}{2}(\bar{e}_1 + \bar{e}_2),\ \bar{l}_2 = \frac{1}{2i}(\bar{e}_1 + \bar{e}_2).$$

У базисі $(\bar{l}_1, \bar{l}_2)$ матриця $A_1 = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \overline{\lambda}_1 \end{bmatrix}$ має вигляд $B_1 = \begin{bmatrix} \alpha_1 & \beta_1 \\ -\beta_1 & \alpha_1 \end{bmatrix}$, де $\lambda_1 = \alpha_1 + i\beta_1,\ \lambda_2 = \alpha_1 - i\beta_1$. Застосувавши таке перетворення до всіх пар власних

векторів з попарно сполученими комплексними власними числами, одержимо представлення лінійного оператора у так званої дійсній жордановій формі:

$$A' = \begin{bmatrix} B_1 & 0 & . & 0 & . & . & 0 \\ 0 & B_2 & . & 0 & . & . & 0 \\ . & . & . & . & . & . & . \\ 0 & ... & 0 & B_k & . & ... & 0 \\ 0 & . & . & 0 & \lambda_{2k+1} & ... & 0 \\ . & . & . & . & . & . & . \\ 0 & . & . & . & 0 & ... & \lambda_n \end{bmatrix}$$

Зауваження. Після переходу до базису з власних векторів коефіцієнти нерівності зміняться. Якщо $S(\Lambda)$ – матриця переходу, то нові значення векторів $\bar{a},\bar{b}$ обчислюються за формулами $\bar{a}^{(S)} = S\bar{a}S^{-1}, \bar{b}^{(S)} = S\bar{b}S^{-1}$. Але для того, щоб не перевантажувати текст новими позначеннями, будемо використовувати старі позначення.

Відзначимо, що матриця виду $\hat{B} \overset{df}{=} \begin{bmatrix} \alpha & \beta \\ -\beta & \alpha \end{bmatrix}$, де $\alpha^2 + \beta^2 = 1$ є матрицею повороту вектора 2-мірного простору на кут φ, заданий співвідношеннями $\cos(\varphi) = \alpha, \sin(\varphi) = \beta$. Тому $B_1 = r_1 \begin{bmatrix} \cos(\varphi_1) & \sin(\varphi_1) \\ -\sin(\varphi_1) & \cos(\varphi_1) \end{bmatrix}$, $r_1 = |\lambda_1| = \sqrt{\alpha^2 + \beta^2}$.

Нерівність (6.25), інваріантність якої розглядається для циклу (6.23) с конкретним початковим значенням $\bar{b}$, означає, що $\forall X \in Orbit(A,\bar{b})(\bar{a}, X) \le (\bar{c},\bar{b})$. Метод доведення інваріантності (6.25) будемо формулювати в еквівалентному виді: $\underset{X \in Orbit(A,\bar{b})}{Sup}(\bar{a}, X) \le (\bar{c},\bar{b})$.

Розглянемо лінійну форму $a_1x_1 + a_2x_2 + ... + a_nx_n \overset{df}{=} (\bar{a}, X)$. Перетворення $X := A^* X$ переводить цю форму в (a, AX), а m – кратна ітерація циклу, визначена цим перетворенням – в $(a, A^m X)$.

Нехай $X_1 = (x_1, x_2), ..., X_k = (x_{2k-1}, x_{2k})$, $\bar{a}_1 = (a_1, a_2), ..., \bar{a}_k = (a_{2k-1}, a_{2k})$. Тоді

$$(\bar{a}, X) = (\bar{a}_1, X_1) + ... + (\bar{a}_k, X_k) + a_{2k+1}x_{2k+1} + ... + a_nx_n \qquad (6.27)$$

Перетворення $(\bar{a}, AX)$ лінійної форми (6.27) можна записати у вигляді

$$(\bar{a}, AX) = (\bar{a}_1, B_1X_1) + ... + (\bar{a}_k, B_kX_k) + \lambda_{2k+1}a_{2k+1}x_{2k+1} + ... + \lambda_n a_n x_n \qquad (6.28)$$

а його m-ту ітерацію – у вигляді

$$(\bar{a}, A^m X) = (\bar{a}_1, B_1^m X_1) + ... + (\bar{a}_k, B_k^m X_k) + \lambda_{2k+1}^m a_{2k+1}x_{2k+1} + ... + \lambda_n^m a_n x_n \qquad (6.29)$$

Перейшовши у (6.29) до представлення у вигляді $B_j = r_j\hat{B}_j$, одержимо:

$$(\bar{a}, A^m X) = r_1^m(\bar{a}_1, \hat{B}_1^m X_1) + ... + r_k^m(\bar{a}_k, \hat{B}_k^m X_k) + \lambda_{2k+1}^m a_{2k+1} x_{2k+1} + ... + \lambda_n^m a_n x_n$$

Розглянемо питання про множину значень орбіти оператора $(\bar{a}_1, \hat{B}_1^m X_1) + ... + (\bar{a}_k, \hat{B}_k^m X_k)$ для початкового значення $\bar{b}^{(0)} = (\bar{b}_1^{(0)}, ..., \bar{b}_k^{(0)})$, де $\bar{b}_j = (b_{2j-1}, b_{2j})$, $j = 1, ..., k$. Будемо розглядати пари X_j як точки на 2-мірній площині, а перетворення $\hat{B}_j \overset{df}{=} \begin{bmatrix} \cos(\varphi_j) & \sin(\varphi_j) \\ -\sin(\varphi_j) & \cos(\varphi_j) \end{bmatrix}$ – як повороти точок X_j на кут φ_j.

Лема 6.1. Нехай

$$B(\varphi) \overset{df}{=} \begin{bmatrix} \cos(\varphi) & \sin(\varphi) \\ -\sin(\varphi) & \cos(\varphi) \end{bmatrix} \tag{6.30}$$

Якщо кут φ є порівняним з π, орбіта $Orbit(B(\varphi), \bar{b}^{(0)})$ є скінченною для будь-якого початкового значення $\bar{b}^{(0)} = (b_1^{(0)}, b_2^{(0)})$, а якщо φ не є порівняним з π, ця орбіта нескінченна й усюди щільна на окружності $x^2 + y^2 = (b_1^{(0)})^2 + (b_2^{(0)})^2$, тобто для будь-якої точки $D(x_0, y_0)$, що лежить на цій окружності, для будь-якого додатного числа ε в ε-околиці точки D існує точка орбіти – послідовності $\{(x^{(j)}, y^{(j)})\}_{j=0}^{\infty}$.

Означення 6.11. Кути φ, ψ назвемо π-еквівалентними, якщо $\varphi - \psi = r\pi$ для деякого $r \in Rat$.

Нехай B_0, B_1 – оператори виду (9) повороту площини на кути φ_0, φ_1 й $Orbit(B_0, X_0), Orbit(B_1, X_1)$ – їх орбіти.

Лема 6.2. Якщо кути φ_0, φ_1 операторів повороту B_0, B_1 не є π-еквівалентними й непорівнянні з π, для будь-яких точок $D_0(x_0, y_0), D_1(x_1, y_1)$, що лежать на одиничній окружності $x^2 + y^2 = 1$, для будь-якого додатного числа ε в ε-околицях точок D_0, D_1 існує по точці орбіт $Orbit(B_0, X_0), Orbit(B_1, X_1)$ з однаковими номерами.

Доведення. Нехай $\psi_0^{(0)}, \psi_1^{(0)}$ – початкові кути орбіт операторів B_0, B_1, ψ_0^*, ψ_1^* – кути точок D_0, D_1. Тоді дія операторів B_0, B_1 визначається формулами:

$$\varphi_0^{(k)} = \psi_0 + k\varphi_0, \ \varphi_1^{(k)} = \psi_1 + k\varphi_1, k = 1, 2, ...$$

Розглянемо різницю

$$\varphi_1^{(k)} - \varphi_0^{(k)} = (\psi_1 - \psi_0) + k(\varphi_1 - \varphi_0), k = 1, 2, ...$$

Через $O(\varepsilon, \varphi)$ позначимо ε-околицю кута φ – тобто відкритий інтервал довжини ε із центром в φ. За лемою 1, для кожного $\varepsilon > 0$ знайдеться таке k_0,

що $\varphi_1^{(k_0)} - \varphi_0^{(k_0)} \in O(\varepsilon/2, \psi_1^* - \psi_0^*)$. Далі, знайдеться таке k_1, що $\varphi_0^{(k_0+k_1)} \in O(\varepsilon/2, \varphi_1^{(k_0)} - \varphi_0^{(k_0)})$. Це означає, що найдеться таке найменше k_2, що $\psi_0^* \in O(\varepsilon/2, \varphi_0^{(k_0+k_2k_1)})$. Тоді $\varphi_0^{(k_0+k_2k_1)} \in O(\varepsilon, \psi_0^*)$, $\varphi_1^{(k_0+k_2k_1)} \in O(\varepsilon, \psi_1^*)$.

Лема 6.3. Якщо кути $\varphi_1,...,\varphi_k$ операторів повороту $B_1,...B_k$ не є попарно π-еквівалентними й непорівнянні з π, для будь-яких точок $D_1(x_1, y_1),...,D_k(x_k, y_k)$, що лежать на одиничній окружності $x^2 + y^2 = 1$, для будь-якого додптного числа ε, в ε-околицях $D_1,...,D_k$ існують точки орбіт $Orbit(B_1, X_1),...,Orbit(B_k, X_k)$ з однаковими номерами.

Лема 6.4. Нехай $(\bar{a}, X) = (\bar{a}_1, X_1) + ... + (\bar{a}_k, X_k)$ – лінійна форма й $B_1,...,B_k$ – двовимірні оператори повороту на кути $\varphi_1,...,\varphi_k$ відповідно, причому $\varphi_1,...,\varphi_k$ не є попарно π-еквівалентними й непорівнянні з π. Визначимо дію оператора A у такий спосіб: $(\bar{a}, AX) = (\bar{a}_1, B_1X_1) + ... + (\bar{a}_k, B_kX_k)$, а початкове значення $\bar{b} = (\bar{b}_1,...,\bar{b}_k)$. Тоді

$$\underset{X \in Orbit(A,\bar{b})}{Sup}((\bar{a}, X)) = \underset{X \in Orbit(B_1,\bar{b}_1)}{Sup}((\bar{a}_1, X_1)) + ... + \underset{X \in Orbit(B_k,\bar{b}_k)}{Sup}((\bar{a}_k, X_k)) \qquad (6.31)$$

Доведення. Для оператора (6.30) розглянемо лінійну форму $(\bar{a}, X) = a_1x + a_2y$. За умовою, аргументи цієї форми визначені на окружності O радіуса $r = \sqrt{b_1^2 + b_2^2}$ із центром у початку координат. Форма $(\bar{a}, X)$ досягає свого максимуму в точці $M(x_M, y_M)$ торкання прямої $a_1x + a_2y = c$ до цієї окружності, причому x_M, y_M й c обчислюються методами аналітичної геометрії. Покладемо $d = \sqrt{a_1^2 + a_2^2}$. Неважко обчислити, що

$$x_M = \frac{a_1 r}{d},\ y_M = \frac{a_2 r}{d},\ \max = a_1x_M + a_2y_M = rd$$

За лемою 6.1, у кожній ε-околиці точки M існують точки орбіти $Orbit(B, \bar{b})$. Тому

$$\underset{X \in Orbit(B_1,\bar{b}_1)}{Sup}((\bar{a}_1, X_1)) = \max_{X \in O_1}(\bar{a}_1, X) = rd.$$

Нехай $M_1,...,M_k$ – точки максимумів для форм $(\bar{a}_1, X_1),...,(\bar{a}_k, X_k)$ на O_1. За лемою 6.3 для системи цих точок і будь-якого додатного ε в ε- околицях цих точок існують точки $D_1,...,D_k$ орбіт операторів $B_1,...,B_k$ з початковими значеннями $\bar{b}_1,...,\bar{b}_k$ відповідно. Тому для обчислення $\underset{X \in Orbit(A,\bar{b})}{Sup}((\bar{a}, X))$ потрібно обчислити $\max_{X \in O_1}(\bar{a}_i, X_i)$ й застосувати (6.31).

Лема 6.5. Нехай лінійна форма $(\bar{a}, X)$, двовимірні оператори $B_1,...,B_k$, оператор A і початкове значення $\bar{b}$ визначені так само, як і в лемі 6.4. Тоді задачу обчислення $\underset{X \in Orbit(A,\bar{b})}{Sup}(\bar{a}, X)$ можна звести до обчислень за формулою (6.31).

Доведення. Нехай $\varphi_1,...,\varphi_l$ – клас еквівалентності φ_1 за відношенням π-еквівалентності $\varphi_i \sim \varphi_j \leftrightarrow \varphi_i - \varphi_j = r_{ij}\pi, r_{ij} \in Rat$. Оскільки точки орбіт усіх операторів B_i лежать на одиничній окружності, можна вважати, що $\begin{bmatrix} x_i \\ y_i \end{bmatrix} = R_i \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$, де R_i – оператори повороту на кути $r_{1i}\pi$. Тому

$(\bar{a}_1, X_1) + ... + (\bar{a}_l, X_l) = (\bar{a}_1 + \bar{a}_2 R_2 + ... + \bar{a}_l R_l) X_1$, а

$(\bar{a}_1, B_1 X_1) + ... + (\bar{a}_l, B_l X_l) = (\bar{a}_1 B_1 + \bar{a}_2 R_2 B_2 + ... + \bar{a}_l R_l B_l) X_1$.

Отже, усі, лінійні форми з π-еквівалентними кутами можна виразити через одну лінійну форму, і у формулі (6.31) можна вважати всі кути попарно нееквівалентними.

Лема 6.6. Нехай $f(x) \in K[x]$ – багаточлен і $\lambda_1,...,\lambda_k$ – комплексне коріння $f(x)$ такі, що $|\lambda_1| = ... = |\lambda_k| = 1$, $\lambda_j = \cos(\varphi_j) + i\sin(\varphi_j), j = 1...k$. Тоді проблема обчислення класів π-еквівалентності на множині $\varphi_1,...,\varphi_k$ алгоритмічно розв'язна.

Доведення. Якщо $\varphi, \psi \in \{\varphi_1,...,\varphi_k\}, \varphi - \psi = r\pi, r \in Rat$, $\lambda = \cos(\varphi) + i\sin(\varphi), \mu = \cos(\psi) + i\sin(\psi)$.

$\lambda / \mu = \cos(\varphi - \psi) + i\sin(\varphi - \psi) = \cos(r\pi) + i\sin(r\pi)$ – корінь деякого ступеня з 1. Алгоритми арифметичних дій над алгебраїчними числами викладені, наприклад, в [71]. Основна ідея полягає у наступному: визначимо систему поліноміальних рівностей

$$(f(u) = 0) \& (f(v) = 0) \& (wv - u = 0)$$

і виключимо з неї змінні u, v методами теорії виключень [19] або базисів Гребнера. Результат – багаточлен $h(w)$ – один з дільників якого повинен бути мінімальним поліномом деякого кореня з 1. Таким чином, $\deg(Gcd(w^d - 1, h(w))) > 0$ для деякого натурального $d \le \deg(h)$.

Лема 6.7. Нехай поліном $f(x) \in K[x]$ і $\lambda_1,...,\lambda_n$ – його корені. Тоді проблема розпізнавання кількості k коренів таких, що $|\lambda_1| = ... = |\lambda_k|$ й $|\lambda_1| = \max(|\lambda_1| = ... = |\lambda_n|)$ алгоритмічно розв'язувана.

Доведення. Покладемо $x = u + iv$. Тоді $f(x)$ можна переписати, виділивши дійсну й уявну частини $f(x)$: $f(u + iv) = re(f(u,v)) + i \cdot im(f(u,v))$. Оскільки $|x|^2 = u^2 + v^2$, задачу визначення чисел з рівними модулями можна сформулювати у вигляді системи рівностей

$$(re(f(u,v)=)0)\ \&\ (im(f(u,v))=0)\ \&\ (w-u^2-v^2=0)\,. \qquad (6.32)$$

Змінна v входить в $im(f(u,v))$ у непарному ступені, тому $im(f(u,v))=v\cdot im(f_1(u,v))$.

Поклавши $v=0$, одержимо $re(f(u,0))=f(u)$. Таким чином, (6.32) спроститься до системи $(f(u)=0)\,\&\,(w-u^2=0)$. Іншими словами, у цьому випадку задача зводиться до відділення й обчислення кратностей дійсних коренів $f(x)$. При $v\neq 0$ система (6.32) спроститься до

$$(re(f(u,v))=0)\ \&\ (im(f_1(u,v))=0)\ \&\ (w-u^2-v^2=0)\,.$$

Оскільки змінна v входить у цю систему в парному ступені, при виключенні змінних алгоритм має справу по суті зі змінної $v_1=v^2$. Виключивши з системи змінні u,v^2, одержимо поліноміальну рівність $g(w)=0,$ дійсні корені якої – суть квадрати модулів комплексної частини чисел $\lambda_1,...,\lambda_n$. Тепер задача зводиться до відділення й визначення кратностей кореня $g(w)=0$. Якщо λ_i – максимальний дійсний корінь $f(x)$, а λ_j – максимальний дійсний корінь $g(x)$ і $\lambda_i=\lambda_j$, $\deg(Gcd(f,g))>0$ і λ_i – дійсний корінь $Gcd(f,g)$. Нехай p – кратність λ_i, а q – кратність λ_j і $\lambda_i=\lambda_j$. Тоді кратність λ_i дорівнює $p+2q$. Якщо $\lambda_i\neq\lambda_j$, кратність шуканого максимального кореня дорівнює або p, або $2q$.

Приклад 6.15. Алгоритми алгебраїчних обчислень задачі доказу інваріантності нерівності.

Нехай $A=\begin{bmatrix}0&1&0\\0&0&1\\2&0&0\end{bmatrix}$.

1. Обчислення характеристичного багаточлена:

$f(x)=|A-xE|=\begin{bmatrix}-x&1&0\\0&-x&1\\2&0&-x\end{bmatrix}=-x^3+2$. $f(x)=x^3-2$. $\lambda_1=\sqrt[3]{2}$, $\lambda_2=\sqrt[3]{2}*\varepsilon$, $\lambda_3=\sqrt[3]{2}*\varepsilon^2$,

$\varepsilon=\cos(\frac{2\pi}{3})+i*\sin(\frac{2\pi}{3})$. ε – первісний корінь ступеня 3 з 1.

2. Алгоритм леми 6.7. Система рівнянь леми:

$f(u+iv)=ref(u,v)+i\cdot imf(u,v)$; $(ref(u,v)=0)\,\&\,(imf(u,v)=0)\,\&\,(w-u^2-v^2=0)$

$x^3-2=0\ \rightarrow\ (u+iv)^3-2=0$

$$\begin{cases}(u^3-3uv^2-2)=0\\ 3u^2v-v^3=0\end{cases}\rightarrow\begin{cases}u^3-3uv^2-2=0\\ 3u^2-v^2=0\end{cases}(1)\ \vee\ \begin{cases}u^3-2=0\\ v=0\end{cases}(2)$$

Випадок 2

$$\begin{cases}u^3-3uv^2-2=0\\ 3u^2-v^2=0\\ w=u^2+v^2\end{cases}\rightarrow\begin{cases}v^2=\dfrac{u^3-2}{3u}\\ 3u^2-v^2=0\\ w=u^2+v^2\end{cases}\rightarrow\begin{cases}uv^2+\dfrac{3}{4}=0\\ 4u^3+1=0\\ w^3-4=0\end{cases}$$

Випадок 1: $q=1$; Випадок 2: p =1. Поліном $f(x)=x^3-2$ має $p+2q=3$ корені з максимальним модулем.

3. Алгоритм леми 6.6.

1.Перевірка порівнянності кута з π. Система: $(f(x)=0)\,\&\,(f(w^2)=0)\,\&\,(x-uw=0)$

$$\begin{cases}x^3-2=0\\ w^6-4=0\\ x-uw=0\end{cases}\rightarrow\begin{cases}x^3-2=0\\ w^3-2=0\\ x-uw=0\end{cases}\rightarrow\begin{cases}(uw)^3-2=0\\ w^3-2=0\\ x-uw=0\end{cases}\rightarrow\begin{cases}u^3-1=0\\ w^3-2=0\\ x-uw=0\end{cases}$$

$\rightarrow \gcd(u^3-1,u^2+u+1)=u^2+u+1\rightarrow\varphi=2\pi/3,$ тобто φ порівняємо з π.

2. Перевірка порівнянності різниці кутів з π. Система:

$$(f(u)=0)\,\&\,(f(v)=0)\,\&\,(wv-u=0)$$

$$\begin{cases}u^3-2=0\\ v^3-2=0\\ wv-u=0\end{cases}\rightarrow\begin{cases}w^3v^3-2=0\\ v^3-2=0\\ wv-u=0\end{cases}\rightarrow\begin{cases}w^3-1=0\\ v^3-2=0\\ wv-u=0\end{cases}\rightarrow\gcd(w^3-1,w^2+w+1)=w^2+w+1\rightarrow$$

$\varphi_u-\varphi_v=2\pi/3,$

тобто $\varphi_u,\varphi_v-\pi$ π- еквівалентні

3. Представлення AX для лінійної форми виду (6.25):

$$A\begin{bmatrix}x\\ y\\ z\end{bmatrix}=(\sqrt[3]{2}\begin{bmatrix}\cos(2\pi/3) & \sin(2\pi/3)\\ -\sin(2\pi/3) & \cos(2\pi/3)\end{bmatrix}\begin{bmatrix}x\\ y\end{bmatrix},\sqrt[3]{2}z)^T$$

4. Обчислення матриці переходу до жорданової форми.

1. Обчислення власних векторів матриці $A=\begin{pmatrix}0&1&0\\0&0&1\\2&0&0\end{pmatrix}$.

Розв'язати систему однорідних лінійних рівнянь $(A-\lambda E)(x,y,z)^T=0$, причому обчислення здійснюються в полі $K(\lambda)$ за модулем λ^3-2.

$\begin{cases}\lambda x-2z=0\\ x-\lambda y=0\\ y-\lambda z=0\end{cases}$. Ранг рівний 2. Фундаментальний розв'язок системи – вектор $s=(\lambda^2,\lambda,1).$ Власні вектори оператора A:

$$s_1=(\lambda_1^2,\lambda_1,1),\ s_2=(\lambda_2^2,\lambda_2,1),\ s_3=(\lambda_3^2,\lambda_3,1).$$

3. Матриця переходу S й зворотна матриця S^{-1}:

$$S=\begin{bmatrix}\lambda_1^2&\lambda_2^2&\lambda_3^2\\ \lambda_1&\lambda_2&\lambda_3\\ 1&1&1\end{bmatrix},\ S^{-1}=\frac{1}{\det(S)}\begin{bmatrix}\lambda_2-\lambda_3&\lambda_2^2-\lambda_3^2&\lambda_2^2\lambda_3-\lambda_2\lambda_3^2\\ \lambda_1-\lambda_3&\lambda_1^2-\lambda_3^2&\lambda_1^2\lambda_3-\lambda_1\lambda_3^2\\ \lambda_1-\lambda_2&\lambda_1^2-\lambda_2^2&\lambda_1^2\lambda_2-\lambda_1\lambda_2^2\end{bmatrix}$$

Теорема 6.10. Проблема доведення інваріантності нерівності $L(\bar a,\bar c,X,\bar b)$ для циклу (6.19) з діагоналізовним лінійним оператором A і даною початковою точкою $\bar b$ алгоритмічно розв'язувана.

Доведення. Оскільки оператор A є діагоналізовним, представлення $(\bar a,A^mX)$ має вигляд:

$$(\bar a,A^mX)=r_1^m(\bar a_1,\hat B_1^mX_1)+\ldots+r_k^m(\bar a_k,\hat B_k^mX_k)+\lambda_{2k+1}^m a_{2k+1}x_{2k+1}+\ldots+\lambda_n^m a_nx_n \qquad (6.33)$$

Нехай p і q - числа, визначені у лемі 6.7 для характеристичного полінома $f(x)$ оператора A. Для визначеності будемо вважати, що власні числа оператора A з максимальними модулями мають початкові номери $\lambda_1,\lambda_2,\ldots\lambda_{2q-1},\lambda_{2q},\lambda_{2q+1},\ldots,\lambda_{2q+p}$, причому $\lambda_1=\overline{\lambda}_2,\ldots\lambda_{2q-1}=\overline{\lambda}_{2q}$. Тоді (6.30) можна представити у вигляді

$$(\bar a,A^mX)=r_1^m(\bar a_1,\hat B_1^mX_1)+\ldots+r_q^m(\bar a_q,\hat B_q^mX_q)+\lambda_{2q+1}^m a_{2q+1}x_{2q+1}+\ldots+\lambda_{2q+p}^m a_{2q+p}x_{2q+p}+L_1^{(m)},$$

де $L_1^{(m)}$ – m-та ітерація лінійної форми виду (6.33) від змінних, що залишилися. Нерівність (6.25) можна переписати у вигляді

$$(\bar a_1,\hat B_1^mX_1)+\ldots+(\bar a_q,\hat B_q^mX_q)+\varepsilon_{2q+1}^m a_{2q+1}x_{2q+1}+\ldots+\varepsilon_{2q+p}^m a_{2q+p}x_{2q+p}+\frac{1}{r_1^m}L_1^{(m)}\le\frac{1}{r_1^m}(\bar c,\bar b)$$

Через $L_0^{(m)}$ позначимо лінійну форму

$$(\bar a_1,\hat B_1^mX_1)+\ldots+(\bar a_q,\hat B_q^mX_q)+\varepsilon_{2q+1}^m a_{2q+1}x_{2k+1}+\ldots+\varepsilon_{2q+p}^m a_{2q+p}x_{2q+p}\ \text{ де }\ \varepsilon_j=\pm1.$$

Одержимо:

$$L_0^{(m)} + \frac{1}{r_1^m} L_1^{(m)} \le \frac{1}{r_1^m} (\bar{c}, \bar{b}). \tag{6.34}$$

Оскільки $r_1 > \max(|\lambda_{2p+q+1}|,...,|\lambda_n|)$, при $m \to \infty$ $\frac{1}{r_1^m} L_1 \to 0$. Отже, для кожного $\delta > 0$ існує таке натуральне m_0, що при $m > m_0$ $\left| \frac{1}{r_1^m} L_1^{(m)} \right| < \delta$.

Мають місце випадки:

1. Якщо $r_1 < 1$, $\lim_{m\to\infty} \frac{1}{r_1^m} (\bar{c}, \bar{b}) = \infty$. Тоді

1.1. при $(\bar{c}, \bar{b}) < 0$ для кожного $\varepsilon > 0$ існує таке m_1 що при $m > m_1$

$$L_0^{(m)} + \frac{1}{r_1^m} L_1^{(m)} < L_0^{(m)} + \delta < -\varepsilon,$$

Тобто при $m > \max(m_0, m_1)$ нерівність $L(\bar{a}, \bar{c}, A^m X, \bar{b})$ не виконується.

1.2. при $(\bar{c}, \bar{b}) > 0$ $L_0^{(m)} + \frac{1}{r_1^m} L_1^{(m)} < L_0^{(m)} + \delta < +\infty$, тобто при $m > \max(m_0, m_1)$ нерівність $L(\bar{a}, \bar{c}, A^m X, \bar{b})$ виконується.

2. Якщо $r_1 = 1$, $\lim_{m\to\infty} \frac{1}{r_1^m} (\bar{c}, \bar{b}) = (\bar{c}, \bar{b})$. При $m > m_0$ $L_0^{(m)} + \frac{1}{r_1^m} L_1^{(m)} < L_0^{(m)} + \delta < (\bar{c}, \bar{b})$. Отже, при $m > m_0$ для виконання нерівності $L(\bar{a}, \bar{c}, A^m X, \bar{b})$ має виконуватися нерівність $L_0^{(m)} < (\bar{c}, \bar{b})$.

3. Якщо $r_1 > 1$, $\lim_{m\to\infty} \frac{1}{r_1^m} (\bar{c}, \bar{b}) = 0$. При $m > m_0$ $L_0^{(m)} + \frac{1}{r_1^m} L_1^{(m)} < L_0^{(m)} + \delta < 0$. Отже, для виконання нерівності $L(\bar{a}, \bar{c}, A^m X, \bar{b})$ при $m > \max(m_0, m_1)$ має виконуватися $L_0^{(m)} < 0$.

Нерівність $L_0^{(m)} < c$ для довільного c на довільній множині S еквівалентна нерівності $\underset{X \in S}{Sup}\, L_0^{(m)} < c$. Тому, незалежно від випадків 1-3, доведення інваріантності зводиться до обчислення

$$\underset{X \in Orbit(A,\bar{b})}{Sup} L_0^{(m)},\ L_0^{(m)} = (\bar{a}_1, \hat{B}_1^m \bar{b}_1) + ... + (\bar{a}_q, \hat{B}_q^m \bar{b}_q) + \varepsilon_{2q+1}^m a_{2q+1} b_{2k+1} + ... + \varepsilon_{2q+p}^m a_{2q+p} b_{2q+p}$$

Алгоритм обчислення $\underset{X \in Orbit(A,\bar{b})}{Sup} (\bar{a}_1, \hat{B}_1^m \bar{b}_1) + ... + (\bar{a}_q, \hat{B}_q^m \bar{b}_q)$ наведено в лемах 6.4, 6.6. Обчислення лінійної частини

$$\underset{X \in Orbit(A,\bar{b})}{Sup} \varepsilon_{2q+1}^m a_{2q+1} b_{2k+1} + ... + \varepsilon_{2q+p}^m a_{2q+p} b_{2q+p}$$

здійснюється безпосередньо розглядом випадків парного й непарного значень m.

Загальний алгоритм доведення інваріантності нерівності $L(\bar{a},\bar{c},X,\bar{b})$ для циклу (6.19) і даного початкового значення $\bar{b}$ полягає у наступному:

1. Привести нерівність до виду (6.34).
2. Обчислити значення $m^* = \max(m_0, m_1)$ й визначити, який з варіантів 1-3 має місце.
3. Перевірити виконання умови інваріантності для певного випадку й $m > m^*$.
4. Перевірити нерівність $L(\bar{a},\bar{c},X,\bar{b})$, виконуючи цикл (6.19) для всіх значень $m \le m^*$.

Приклад 6.16. (Продовження прикладу 6.14)
Нагадаємо, що

$$S(b_1,b_2) = (0 \le b_1 \le 1)\,\&\,(0 \le b_2 \le 1),\ U(x,y,b_1,b_2) = \neg(|x+b_1| \le \varepsilon)\,\&\,(|y+b_2| \le \varepsilon),$$

$$A = \begin{bmatrix} 3/5 & 4/5 \\ -4/5 & 3/5 \end{bmatrix},\ L = x + y \le 2b_1 + 2b_2\ \ //\,\bar{a} = (1,1), \bar{c} = (2,2)\ .$$

1. Обчислити характеристичний поліном

$$A = \begin{bmatrix} 3/5 & 4/5 \\ -4/5 & 3/5 \end{bmatrix}\quad |A - xE| = \begin{vmatrix} 3/5 - x & 4/5 \\ -4/5 & 3/5 - x \end{vmatrix} = x^2 - 6/5x + 1.$$

2. Знайти кількість коренів із максимальним модулем:

$$x^2 - 6/5x + 1 = 0 \ \rightarrow\ (u+iv)^2 - 6/5(u+iv) + 1 = 0$$

$$\begin{cases} u^2 - v^2 + 6/5u + 1 = 0 \\ 2uv - 6/5v = 0 \end{cases} \rightarrow \begin{cases} u^2 - v^2 + 6/5u + 1 = 0 \\ u - 3/5 = 0 \end{cases} \text{(вип.1)} \vee \begin{cases} u^2 + 6/5u + 1 = 0 \\ v = 0 \end{cases}$$

(вип.2)

Випадок 1 тривіальний. Обчислимо випадок 2:

$$\begin{cases} u^2 - v^2 + 6/5u + 1 = 0 \\ u - 3/5 = 0 \\ w = u^2 + v^2 \end{cases} \rightarrow \begin{cases} 9/25 - v^2 + 18/25 + 1 = 0 \\ u - 3/5 = 0 \\ w = u^2 + v^2 \end{cases} \rightarrow \begin{cases} v^2 - 16/25 = 0 \\ u - 3/5 = 0 \\ w = 1 \end{cases}$$

Таким чином, випадок 1: $q = 0$; випадок 2: p =1. $f(x) = x^2 - 6/5x + 1$ має $p + 2q = 2$ комплексних кореня з максимальним модулем, рівним 1.

3. Перевірити порівнянність кута з π: $(f(x) = 0)\,\&\,(f(w^2) = 0)\,\&\,(x - uw = 0)$

$$\begin{cases} x^2 - 6/5x + 1 = 0 \\ w^2 - 1 = 0 \\ x - uw = 0 \end{cases} \rightarrow \begin{cases} x^2 - 6/5x + 1 = 0 \\ w - 1 = 0 \\ x - u = 0 \end{cases} \rightarrow \begin{cases} u^2 - 6/5u + 1 = 0 = 0 \\ w - 1 = 0 \\ x - u = 0 \end{cases}$$

$\rightarrow \gcd(u^2 - 6/5u + 1 = 0, u - 1) = 1 \rightarrow$ т.е. φ непорівнянний з π

4. Перевірити порівнянність різниці кутів з π: $(f(u) = 0) \& (f(v) = 0) \& (wv - u = 0)$

$$\begin{cases} u^2 - 6/5u + 1 = 0 \\ v^2 - 6/5v + 1 = 0 \\ wv - u = 0 \end{cases} \rightarrow \begin{cases} v = (w+1)/(6/5w) \\ v^2 - 6/5v + 1 = 0 = 0 \\ wv - u = 0 \end{cases} \rightarrow \begin{cases} v = (w+1)/(6/5w) \\ 25w^2 + 14w + 25 = 0 \\ wv - u = 0 \end{cases}$$

$\rightarrow \gcd(25w^2 + 14w + 25, w^2 - 1) = 1 \rightarrow$ т.е. φ_u, φ_v не є π-еквівалентними.

5. Представити A в дійсній жордановій формі. A вже представлена в необхідній формі, тому $S = E$.

6. Обчислити за лемою 6.4 $\underset{(x,y)\in Orbit(A,\bar{b})}{Sup} (\bar{a}, (x,y)^T) = \sqrt{2} \cdot \sqrt{b_1^2 + b_2^2}$.

7. Оскільки $|\lambda| = w = 1$, має місце випадок 2 теореми 6.9.

8. За теоремою 6.10 нерівність $(\bar{a}, X) \le (\bar{c}, \bar{b})$ має виконуватися для усіх вершин многокутника предумови $S(\bar{b})$, тобто, за лемою 6.4

$$\underset{(x,y)\in O(b_1,b_2)}{Sup} (x + y) \le b_1 + b_2 \text{ для } (b_1,b_2) = (0,0), (b_1,b_2) = (0,1), (b_1,b_2) = (1,0), (b_1,b_2) = (1,1).$$

Перевірка: $\sqrt{2} \cdot 0 \le 0$, $\sqrt{2} \cdot \sqrt{b_1^2 + b_2^2} = \sqrt{2} \le 2$, $\sqrt{2} \cdot \sqrt{b_1^2 + b_2^2} = \sqrt{2} \le 2$, $\sqrt{2} \cdot \sqrt{b_1^2 + b_2^2} = 2 \le 4$.

Таким чином, нерівність $(\bar{a}, X) \le (\bar{c}, \bar{b})$ виконується для всіх $\bar{b} \in S(\bar{b})$.

Оскільки дійсна жорданова форма складається з однієї 2×2 клітки, перехід до межі в теоремі 6.10 і обчислення $m*$ не потрібно.

Теорема 6.11. Проблема доведення інваріантності нерівності $L(\bar{a}, \bar{c}, X, \bar{b})$ для циклу (6.19) з даною початковою точкою $\bar{b}$ є алгоритмічно розв'язуваною.

Доведення полягає в повторенні міркувань теореми 6.10 на випадок матриці A з нетривіальними жордановими клітинами. Нехай матриця оператора A у жордановій формі має нетривіальні жорданові клітини. Оскільки для кожного комплексного власного значення λ існує сполучене власне значення $\bar{\lambda}$, жорданова форма A разом із клітиною $J_k(\lambda)$ містить клітину $J_k(\bar{\lambda})$. Підматриця дійсної жорданової форми оператора A, відповідна до пари $(\lambda, \bar{\lambda})$, має вигляд

$$J(\lambda)=\begin{bmatrix} B(\varphi) & E & ... & 0 \\ 0 & B(\varphi) & ... & 0 \\ . & . & . & . \\ 0 & ... & B(\varphi) & E \\ 0 & ... & 0 & B(\varphi) \end{bmatrix}, B(\varphi)=r\begin{bmatrix} \cos(\varphi) & \sin(\varphi) \\ -\sin(\varphi) & \cos(\varphi) \end{bmatrix}, E=\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

і матриця оператора A має вигляд $A=J_1(\lambda_1)\times...\times J_l(\lambda_l)\times J_{l+1}(\lambda_{l+1})\times J_n(\lambda_{l+k})$, де $\lambda_1,\overline{\lambda}_1...\lambda_l,\overline{\lambda}_l$ – комплексні власні числа, а $\lambda_{l+1},\lambda_{l+k}$ – дійсні власні числа A.

$$A^m=J_1^m(\lambda_1)\times...\times J_l^m(\lambda_l)\times J_{l+1}^m(\lambda_{l+1})\times J_{l+k}^m(\lambda_{l+k}). \qquad (6.32)$$

Кожній клітині $J_j(\lambda_j)$ відповідає своя група змінних. Розглянемо тому довільну жорданову клітину розміру k, позначену в (6.32). Елементи цієї клітини – 2×2 матриці. Нехай $X=(X_1,...,X_k), X_j=(x_{2j-1},x_{2j})$. Пара (x_{2j-1},x_{2j}) відповідає парі сполучених власних чисел $(\lambda_j,\overline{\lambda}_j)$. Нехай m – натуральне число й $\overline{b}=(\overline{b}_1,...,\overline{b}_k), \overline{b}_j=(b_{2j-1},b_{2j})$ – вектор пар змінних – початкових значень. Обчислимо m-ту ітерацію $X^{(0)}=\overline{b}$; $X^{(m)}=AX^{(m)}$ у явному виді $X^{(m)}=A^mX$: $X^{(m)}=J(\lambda)^m\overline{b}$. Одержимо:

$$X^{(m)}=\begin{bmatrix} r^mB(m\varphi) & C_1(m) & ... & C_{k-1}(m) \\ 0 & r^mB(m\varphi) & ... & C_{k-2}(m) \\ 0 & ... & r^mB(m\varphi) & C_1(m) \\ 0 & ... & 0 & r^mB(m\varphi) \end{bmatrix}\begin{bmatrix} \overline{b}_1 \\ \overline{b}_2 \\ ... \\ \overline{b}_k \end{bmatrix}, b_j=\begin{bmatrix} b_{2j-1} \\ b_{2j} \end{bmatrix}. \qquad (6.36)$$

де $C_j(m)=C_m^jr^{m-j}B((m-j)\varphi)=\dfrac{m(m-1)...(m-j+1)}{j!}r^{m-j}B((m-j)\varphi)$, $j\in 1..k-1$.

Рівність, що відповідає j-тому рядку (6.36), має вигляд:

$$x_j^{(m)}=r^m(\overline{b}_jB(m\varphi)+\frac{C_1(m)}{r}\overline{b}_{j+1}+...+\frac{C_{k-j}(m)}{r^{k-j}}\overline{b}_k)\overset{df}{=}r^mg_j(m,r,b). \qquad (6.37)$$

Лінійна форма – аналог (6.27) є сумою, кожний доданок якої визначено групою змінних клітини $J_j(\lambda_i)$:

$$r_j^mS_j^{(m)}=r_j^m(g_1(m,r_j,b_{j1})a_{j1}+g_2(m,r_j,b_{j2})a_{j2}+...+g_{k_j}(m,r_j,b_{jk_j})a_{jk_j}). \qquad (6.38)$$

Аналог лінійної нерівності (6.25) має вигляд

$$r_1^mS_1^{(m)}+r_2^mS_2^{(m)}+...+r_l^mS_l^{(m)}\le c \qquad (6.39)$$

Позначимо через r_1 максимальне значення: $r_1=\max(|\lambda_1|,...,|\lambda_l|)$. Тоді, оскільки $S_j^{(m)}$ – багаточлен від m, а $r_j/r_1)^m$ – показова функція від m і $r_j/r_1<1$,

$$\frac{r_j^m S_j^{(m)}}{r_1^m} \underset{m\to\infty}{\to} 0 \quad \text{при } r_j \neq r_1 . \; \frac{r_1^m S_1^{(m)}}{r_1^m} \underset{m\to\infty}{\to} \bar{b} B_1(m\varphi) .$$

Тому метод теореми 6.10 може бути використаний і в даній теоремі, тобто в загальному випадку.

Теорема 6.12. Проблема доказу інваріантності нерівності $L(\bar{a},\bar{c},X,\bar{b})$ для циклу (6.19) (тобто з даною передумовою $S(b)$) алгоритмічно розв'язувана.

*Доведення***.** Лінійну напівалгебраїчну множину $S=\{\bar{b} \mid S(\bar{b}\}$ можна представити у вигляді об'єднання опуклих багатогранних множин, оскільки формулу $S(b)$ можна представити у вигляді $S(b)=S_1(b)\vee ...S_l(b)$, де $S_1(b)\vee ...S_l(b)$ – системи лінійних нерівностей. В ([107, леми 2, 3]) доведено, що виконання лінійної нерівності для будь-якої точки опуклого багатогранника випливає з його виконання для усіх вершин цього багатогранника. Тому доведення інваріантності $L(\bar{a},\bar{c},A^m X,\bar{b})$ на $S(b)$ полягає в його перевірці для всіх $S_1(b),...,S_l(b)$.

Теорема 6.13. Проблема завершеності циклу (6.22) алгоритмічно розв'язувана.
Доведення. Цикл (6.22) розходиться тоді й тільки тоді, коли умова $U(x,b)-$ інваріант циклу (6.23).

6.6. Висновки

1. Аналіз задачі генерації програмних інваріантів у вигляді поліноміальних рівностей показує, що основним методом комп'ютерної алгебри, який використовується, є метод базисів Гребнера. Дійсно, множина поліноміальних інваріантів утворює ідеал кільця поліномів від змінних комп'ютерної програми, що аналізується. Основний результат – теорема 6.2 та метод невизначених коефіцієнтів обчислення інваріантів, показаний у прикладі 6.5.

2. Розв'язуваність задачі генерації поліноміальних інваріантів простого лінійного циклу обумовлена, як це стверджує теорема 6.6, розв'язуваністю співвідношення (6.20). Задача повного аналізу множини розв'язків цього співвідношення є відкритою.

3. Задачу доведення інваріантності лінійної нерівності розв'язано повністю.

Література

Гл 1

1. Gardner M. A New Kind of Cipher that Would Take Millions of Years to Break / M. Gardner. // Scientific American. – 1977. – Iss. 37. – P. 120–126.
2. Rivest R. A Method for Obtaining Digital Signatures and Public-key Cryptosystems / R. Rivest, A. Shamir, L. Adleman. // Communications of the ACM. – 1978. – Vol. 21, Iss. 2. – P. 120–126.
3. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си / Б. Шнайер. – Москва: Триумф, 2002. – 816 с.
4. Курош А. Г. Лекции по общей алгебре / А. Г. Курош. – Москва: Наука, 1973. – 400 с.
5. Мальцев А. И. Алгебраические системы / А. И. Мальцев. – Москва: Наука, 1970. – 370 с.
6. Кон П. Универсальная алгебра / П. Кон. – Москва: Мир, 1968. – 348 с.
7. Ван дер Варден Б.Л. Алгебра / Б.Л. Ван дер Варден; изд. 2-ое. – М: ГРФМЛ, 1979. – 624 с.
8. Ленг С. Алгебра / С. Ленг. – Москва: Мир, 1968. – 564 с.
9. Клиффорд А. Алгебраическая теория полугрупп. Том 1. / А. Клиффорд, Г. Престон. – Москва: Мир, 1972. – 283 с.
10. Клиффорд А. Алгебраическая теория полугрупп. Том 2. / А. Клиффорд, Г. Престон. – Москва: Мир, 1972. – 422 с.
11. Курош А. Г. Теория групп / А. Г. Курош. – 3-е изд.– Москва: Физматлит, 1967. – 648 с.
12. Холл М. Теория групп / М. Холл. – Москва: ИЛ, 1962. – 460 с.
13. Зарисский О. Коммутативная алгебра. Том 1. / О. Зарисский, П. Самюэль. – Москва: ИЛ, 1963. – 373 с.

14. Зарисский О. Коммутативная алгебра. Том 2. / О. Зарисский, П. Самюэль. – Москва: ИЛ, 1963. – 438 с.

15. Джекобсон Н. Строение колец / Н. Джекобсон. – Москва: ИЛ, 1961. – 376 с.

16. Сикорский Р. Булевы алгебры / Р. Сикорский. – Москва: Мир, 1969. – 376 с.

17. Постников М. М. Теория Галуа / М. М. Постников. – Москва: Физматлит, 1963. – 220 с.

18. Феферман С. Числовые системы. Основания алгебры и анализа / С. Феферман. – Москва: Наука, 1971. – 440 с.

19. Ходж В. Методи алгебраїчної геометрії / В. Ходж, Д. Пидо. – Москва: МУЛ, 1954. – 462 с.

20. Бухштаб А. А. Теория чисел / А. А. Бухштаб. – Санкт-Петербург: Лань, 2015. – 384 с.

21. Виноградов И. М. Основы теории чисел / И. М. Виноградов. – Москва: Гостехиздат, 1952. – 180 с.

22. Артин Э. Теория Галуа. / Пер. с англ. А. В. Самохина. – 2-е изд. стереотипное. – М.: МЦНМО, 2008. – 66 с. – (Классические монографии: математика).

Гл 2.

23. Dershowitz N. Rewrite Systems. Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics. / N. Dershowitz, J.P. Jouannaud. – Elsevier and MIT Press, 1990. – P. 243-320.

24. Terese U. Term Rewriting Systems. / U. Terese // Cambridge Tracts in Theoretical Computer Science, №55. – Cambridge: Cambridge University Press, 2003. – 908 p.

25. Baader F. Term Rewriting and All That / F. Baader, T. Nipkow. – Cambridge: Cambridge University Press, 1999. – 316 с.

26. Kapitonova J.V. Algebraic programming in APS system / J.V. Kapitonova, A.A. Letichevsky, S.V. Konozenko // In Proc. of the Int. Symp. on Symbolic and Algebraic Computation (ISSAC`90). – New York: ACM, 1990. – P. 68-75.

27. Letichevsky A.A. Algebraic Programming System APS: User Manual / A.A. Letichevsky, J. Kapitonova, V. Volkov and others // Glushkov Institute of Cybernetics, National Acad. of Sciences of Ukraine, Kiev, Ukraine, 1998. – 50 p.

28. Капитонова Ю.В. Дедуктивные средства системы алгебраического программирования / Ю.В. Капитонова, А.А. Летичевский, В.А. Волков // Кибернетика и системный анализ. – 2000. – №1. – С. 17–35.

29. Lvov M. Tools for Solving Problems in the Scope of Algebraic Programming. / J.Kapitonova, A.Letichevsky, M. Lvov, V. Volkov// Lecture Notes in Computer Sciences. –№ 958. – 1995. – P. 31-46.

30. Капитонова Ю.В. Разработка решателей задач в АПС / Ю.В. Капитонова, А.А. Летичевский, В.А. Волков // Тр. Междунар. конф. “Интеллектуальные системы и компьютерное моделирование”. – М ., 1995. – С.62–71.

31. Letichevsky A.A. The Algebraic Programming System for Parallel Symbolic Simulation / A.A. Letichevsky, Y.V. Kapitonova, A.E. Doroshenko, V.A. Volkov // Proc. Intern. Conf. on Algebraic Eng. – Aizu (Japan): World Sci. Publisher, March 24-28, 1997. – P. 350– 360.

32. Денисенко П.М. Оптимальна апроксимація спеціальних функцій в системі алгебраїчного програмування / П.М. Денисенко, О.А. Летичевський, В.А. Волков // Теорія обчислень. – К.: Ін-т кібернетики ім. В.М.Глушкова НАН України, 1999. – С. 146–150.

33. Капитонова Ю.В. Дедуктивные средства системы алгебраического программирования / Ю. В. Капитонова, А.А. Летичевский, В.А. Волков // Кибернетика и системный анализ. – 2000. – № 1. – С.17-35.

34. Песчаненко В.С. Использование системы алгебраического программирования APS для построения систем поддержки изучения

алгебры в школе / В.С. Песчаненко // Управляющие системы и машины. – 2006. – №4. – С. 86-94.

35. Песчаненко В.С. Розширення стандартних модулів системи алгебраїчного програмування APS для використання у системах навчального призначення / В.С. Песчаненко // Науковий часопис НПУ імені М.П. Драгоманова: зб. наук.праць. – К.:НПУ ім. М.П. Драгоманова, 2005. – № 3 (10). – С. 206–215. – (Серія «Комп'ютерно-орієнтовані системи навчання»).

Гл 3

36. Глушков В.М. Алгебра. Языки. Программирование / В.М. Глушков, Г.Е. Цейтлин, Е.Л. Ющенко. – Киев: Наук. думка, 1989. – 376 с.

37. Математическая логика в программировании: [сб.статей 1980-1988 гг.]. – М.:Мир, 1991. – 408 с.

38. Guttag J.V. The algebraic specifications of abstract data types / J.V.Guttag, J.J. Horning // Acta informatica. – 1978. – №10. – P. 27-52.

39. Goguen J. Ordered-Sorted Algebra I: Partial and Overloaded Operations. Errors and Inheritance. / J. Goguen, J. Meseguer // Theoretical Computer Science. – Oxford: Elsevier. – 1992. – Vol. 105 (Num 2). – P. 217-273.

40. Goguen J.A. An initial algebra approach to the specification, correctness and implementation of abstract data types / J.A.Goguen, J.W. Thatcher, E. Wagner // Current Trends in Programming Methodology. – [R.Yeh eds.]. – Englewood Cliffs, NJ: Prentice Hall, 1978. – P. 80-149.

41. Meseguer J. Inintialiy, Induction and Computerbility / J. Meseguer, J. Goguen // Algebraic Methods in Semantics, [Nivat M., Reingolds C. eds.]. – Cambridge University Press, 1985. – P. 459-541.

42. Goguen J.A. Eqlog: Equality, Types and generic Modules for Logic Programming. / J.A. Goguen, J. Meseguer // Functional and Logic

Programming. – [Douglas DeGroot, Gary Lindstrom, Prentice Hall eds.]. – 1986, P. 295-363.

43. Principles of OBJ-2.-Proc / Futatsugi K., Goguen J.A., Jouannaud J.P., Meseguer J. // 12th ACM Symposium on Principles of Programming Languages. – [ed. B.Reid]. – ACM,1985. – P. 52-66.

44. Агафонов В.Н. Языки и средства спецификации программ / В.Н. Агафонов // Требования и спецификации в разработке программ. – М.: Мир, 1984. – С.285-344.

45. Ehrig Y. Fundamentals of algebraic specifications Vol. 1: Equations and inintial semantics. / Y. Ehrig, B. Mahr // EATCS Monographts on theoretical Computer Sciencs Vol.6. – [eds. W.Broure, G.Rosenberg, A.Salomaa]. – Springer-Verlag, 1985. – 192 p.

46. Danforth S. Type theories and object-oriented programming / S. Danforth, M. Reeve // ACM Computing Surveys. – 1988. – v.20, № 1. – P 29-72.

47. The Scientific Computation System. [Електронний ресурс] // Axiom™. – Режим доступу: http://www.axiom-developer.org/.

48. The Scientific Computation System. Books. [Електронний ресурс] // Axiom™. – Режим доступу: http://www.axiom-developer.org/axiom-website/books.html.

49. Львов М.С. Синтез інтерпретаторів алгебраїчних операцій в розширеннях багатосортних алгебр /М.С. Львов // Вісник Харьківського національного университету. – 2009. – №847. – С.221-238. – (Серія «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління»).

50. Львов М.С. Об одном подходе к реализации алгебраических вычислений: вычисления в алгебре высказываний / М.С.Львов // Вестник Харк. нац. ун-та. – 2009. – № 863. – С. 157-168. – (Серия "Математическое моделирование.

Информационные технологии. Автоматизированные системы управления").

51. Песчаненко В.С. Об одном подходе к проектированию алгебраических типов данных / В.С. Песчаненко // Проблемы программирования. – 2006. – №2-3. – С. 626-634.

52. Львов М.С. Об одном подходе к верификации алгебраических вычислений. / М.С.Львов // Проблемы программирования. – 2011. – № 4. – С. 23-35.

53. Bouhoula A. Automated Mathematical Induction / A. Bouhoula, E. Kounalis, M. Rusinowitch // INRIA. Rapports de Recherhe. – 1992. – №1663. – 44 p .

54. Bouhoula A. Implicit Induction in Conditional Theories / A. Bouhoula, M. Rusinowitch // INRIA. Rapports de Recherhe. – 1993. – №2045. – 39 p .

55. Musser D.R. On proving inductive properties of abstract data types / D.R. Musser // Proc. Of 7th ACM Symp. On principles of programming languages. – Association fof Computing Machinery. – 1980. – P. 154-162.

56. Reddy U.S. Term Rewriting Induction / U.S. Reddy // Proc. 10th Int.Conf. on Automated Deduction, Lecture Notes in Computer Science. – Springer-Verlage. – [M.E.Sticel eds.]. – 1990. – vol. 449. – P. 162-177.

57. Львов М.С. Метод спадкування при реалізації алгебраїчних обчислень в математичних системах навчального призначення / М.С.Львов// Системи управління, навігації та зв'язку. – К: ЦНДІ НіУ, 2009. – Вип. 3 (11). – С.120-130.

58. Львов М.С. Метод морфізмів реалізації алгебраїчних обчислень в математичних системах навчального призначення / М.С.Львов // Системи обробки інформації. – Харків: ХУПС, 2009. – Вип.6 (80). – С.183-190.

59. Тан К. Символьный С ++: Введение в компьютерную алгебру с использованием объектно-ориентированного программирования / К. Тан, В. Стиб, Й. Харди – М.:Мир, 2001 – 624 с.

60. Пол А. Объектно-ориентированное программирование на C++ / А. Пол. – [2-е изд.]. – М.: «БИНОМ», 1999. – 462 с.

61. Goguen J.A. Parameterized programming. – IEEE Transact / J.A. Goguen // On Soft Engineering, SE-10. – 1984. – № 5. – P. 528-543.

62. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++ / Г. Буч. – М.: «Бином», 1998. – 560 с.

63. Акритас А. Основы компьютерной алгебры с приложениями / А. Акритас. – М.: Мир, 1994. – 543 с.

Гл 4

64. Сканави М.И. Сборник задач по математике для поступающих в ВУЗы / М.И. Сканави. – М.:Оникс, 2009. – 608 с.

65. Подколзин А.С. Компьютерное моделирование логических процессов. Том 1. Архитектура и языки решателя задач. / А.С. Подколзин. – М.: Физматлит, 2008. – 1024 с.

Гл. 5

66. Карацуба А. Умножение многозначных чисел на автоматах /А. Карацуба, Ю. Офман // Доклады Академии Наук СССР. – 1962. – Т. 145. – № 2.

67. Schönhage A. Schnelle Multiplikation großer Zahlen / A. Schönhage, V. Strassen // Computing. – 1971. – № 7. – P. 281-292.

68. Кнут Д. Искусство программирования. Том 2. Получисленные алгоритмы, 3-е изд. – М.: Издательский дом «Вильямс», 2001. – 832 с.

69. Дэвенпорт Дж. Компьютерная алгебра. / Дж. Дэвенпорт, И. Сирэ, Э. Турнье. – М.: Мир, 1991. – 352 с.

70. Акритас А.Г. Основы компьютерной алгебры с приложениями / А.Г. Акритас. – М.: Мир, 1994. – 272 с.

71. Комп'ютерна алгебра: символьні й алгебраїчні обчислення. – [ред. Б.Бухбергера, Дж.Коллинза, Р.Лооса.]. – М.:Мир, 1986. – 392 с.

72. Проасолов В. В. Многочлены. / В. В. Проасолов. – М.: МЦНМО, 2014. – 335 с.
73. Кокс Д. Идеалы, многообразия и алгоритмы. / Д. Кокс, Дж. Литтл, Д. О'Ши. – М.: Мир, 2000. – 687 с.
74. Верещагин Н.К. Лекции по математической логике и теории алгоритмов. Ч.2 Языки и исчисления. / Н.К. Верещагин, А. Шень. – М.: МЦНМО, 2012. – 240 с.
75. Барвайс Дж. Справочная книга по математической логике: В 4-х частях. Ч. IV. Теория доказательств и конструктивная математика./ Дж. Барвайс. – М.: Наука, 1983. – 392 с.
76. Моцкин Т.С. Метод двойного описания. / Т.С. Моцкин, Х. Райфа, Дж.Л. Томпсон, Р.М. Тролл // Матричные игры. – М.:Физматлит, 1961. – С.81–109.
77. Ziegler G.M. Lectures on polytopes: Graduate Texts in Mathematics / Ziegler G.M. – New York:Springer-Verlag, 1995. – 370 p.
78. Черников С.Н. Линейные неравенства. / С.Н. Черников. – М.: Наука, 1968. – 490 с.
79. Солодовников А.С. Системы линейных неравенств. / А.С. Солодовников. – М.:Наука, 1977. – 112 с.
80. Емеличев В.А. Многогранники, графы, оптимизация / В.А. Емеличев, М.М. Ковалев, М.К. Кравцов. – М.: Наука, 1981. – 348 с.
81. Артамонов В.А. Линейная алгебра и выпуклая геометрия / В.А. Артамонов, В.Н. Латышев. – М.: Изд-во «Факториал Пресс», 2004. – 160 с.
82. Чарин В. С. Линейные преобразования и выпуклые множества / В.С.Чарин. – Киев: Высшая школа, 1978. – 192 с.
83. Препарата Ф. Вычислительная геометрия: введение. / Ф. Препарата, М. Шеймос. – М.:Мир, 1989. – 478 с.
84. Ласло М. Вычислительная геометрия и компьютерная графика на С++ / М. Ласло. – М.: «Бином», 1997. – 304 с.

Гл 6.

85. Ершов А.П. Введение в теоретическое программирование: беседы о методе. / А.П. Ершов. – М.: Наука, 1977. – 288 с.

86. Основи дискретної математики / Ю.В.Капітонова, С.Л. Кривий, О.А.Летичевський та ін. – К.: Наукова думка, 2002. – 580 с.

87. Львов С. М. О реализации вычислений в задачах анализа программ, определенных над векторными пространствами / С.М. Львов // Проблемы программирования. – 2004. – №2-3. – С. 95–101. – (Спецвыпуск).

88. Львов С.М. О технологиях построения и обработки математических моделей программ / С.М. Львов // Проблемы программирования. – 2007. – №3. – С. 41-48.

89. Floyd R. W. Assigning Meanings to Programs. / R. W. Floyd // in Proceedings of Symposium on Applied Mathematics. Providence, R.I.: American Mathematical Society. – 1967. – Vol. 19. – P. 19-32.

90. Hoare A.R. Anaxiomatic basis for computer programming. / A.R. Hoare // Communications of the ACM. – NewYork: ACM, 1969. – №12(10). – P.576-580.

91. Godlevsky A.B. Iterative methods of program analysis. / A.B. Godlevsky, J.V. Kapitonova, S.L. Krivoy, A.A. Letichevsky // Cybernetics. – 1989. – №2. – P .9-19.

92. Львов М.С. Інваріантні рівності малих ступенів у програмах, визначених над полем. / М.С. Львов // Кібернетика. – 1988. – № 1. – Р. 108-110.

93. Letichevsky A. Discovery of Invariant Equalities in Programs over Data Fields. / A. Letichevsky, M. Lvov // Applicable Algebra in Engineering, Communication and Computing. – 1993. – №4. – P. 21-29.

94. Lvov M. About one algorithm of program polynomial invariants generation. / M. Giese, T. Jebelean. (Eds.)// Proc. Workshop on Invariant Generation, WING

2007. Technical report №07-07 in RISC Report Series, University of Linz, Austria. 06 2007. Workshop Proceedings. – P. 85-99.

95. Müller-Olm M. Computing polynomial program invariants. / M. Müller-Olm, H. Seidl // Information Processing Letters. – Vol 91, Issue 5. – Elsevier North-Holland, Inc. Amsterdam. – P. 233-244.

96. Müller-Olm M. Precise Interprocedural Analisys through Linear Algebra. / M. Müller-Olm, H. Seidl // Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages. – Vol.39, Issue 1. – New York: ACM. – P. 330-341.

97. Sankaranarayanan S. Non-linear loop invariant generation using Gröbner bases./ S. Sankaranarayanan, H. Sipma, Z. Manna // Proc. of Symposium on Principles of Programming Languages – Venice, Italy, January 14-16, 2004. – New York : ACM, 2004. – P. 318-329.

98. Rodríguez-Carbonell E. Automatic generation of polynomial loop invariants: algebraic foundations. / E. Rodríguez-Carbonell, D. Kapur // Proc. Of International Symposium on Symbolic and Algebraic Computation – Santander, Spain, July 4-7, 2004. – New York: ACM, 2004. – P. 266-273.

99. Kovács L. I. An Algorithm for Automated Generation of Invariants for Loops with Conditionals. / L. I. Kovács, T. Jebelean // Proc. of Intern. Symp. on Symbolic and Numeric Algorithms for Scientific Computing – Timisoara, Romania, 25-29 Sept. 2005. – IEEE Computer Society, 2005. – P. 245-249.

100. Lvov M.S. Polynomial invariants for linear loops. / M.S. Lvov // Cybernetics and Systems Analysis. – Vol. 46, Issue 4. – 2010. – P. 660-668.

101. Львов М.С. Нелінійні інваріанти лінійних циклів і власні поліноми лінійних операторів. / М.С. Львов, В.А. Крекнин // Кібернетика й системний аналіз. – 2012. – № 2. – С. 126-139.

102. Cousot P. Automatic discovery of linear restraints among variables of a program. / P. Cousot, N. Halbwachs // in Conference Record of the Fifth Annual

ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. – New York: ACM, 1978. – P. 84-97.

103. Кривий С.Л. Пошук інваріантних лінійних залежностей у програмах. / С.Л. Кривий, С.Г. Ракша // Кібернетика. – 1984. – № 6. – С. 23-28.

104. Годлевский А.Б. Ітеративні методи аналізу програм. Рівності й нерівності. / А.Б. Годлевский, Ю.В. Капітонова, С.Л. Кривій, А.А. Летичевский // Кібернетика. – 1990. – № 3. – С. 1-10.

105. Львов М.С. Інваріантні нерівності в програмах, інтерпретованих над упорядкованими полями. / М.С. Львов // Кібернетика. – 1986. – №5. – С.22-27.

106. Львов М.С. Про інваріантні нерівності для станів схем програм, інтерпретованих над векторним простором. / М.С. Львов // Кібернетика. – 1985. – №2. – С.111-112.

107. Львов М.С. Метод доведення інваріантності лінійних нерівностей для лінійних циклів. / М.С. Львов // Кібернетика й системний аналіз. – 2014. – № 4. – С. 80-85.

108. Робинсон А. Введення в теорію моделей і метаматематику алгебри. / А. Робинсон. – М.:"Наука", ГЗФМЛ. – 1967. – 376 с.

109. Ахо А. Построение и анализ вычислительных алгоритмов. / А. Ахо, Дж. Хопкрофт, Дж. Ульман. – М.: Мир, 1979. – 536 с.