Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережних технологій факультету інформатики

Тензорні обчислення в системі комп'ютерної алгебри

Текстова частина до дипломної роботи
за спеціальністю „Інженерія програмного забезпечення" 121

Керівник дипломної роботи
д. ф.-м.н., пр. Малашонок Г.І.
(прізвище та ініціали)

_____
(підпис)
"16" червня 2020 р.

Виконав студент 2 курсу
магістерської програми
Вороняк О.І.
(прізвище та ініціали)
"16" червня 2020 р.

Київ 2020

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережних технологій факультету інформатики

ЗАТВЕРДЖУЮ
Зав.кафедри мережних технологій,
д. ф.-м.н., пр _____Г. І. Малашонок

«____»_____2019р

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на дипломну роботу
студенту ***Вороняку Остапу Ігоровичу*** факультету ***інформатики 2*** курсу магістерської програми

ТЕМА Тензорні обчислення в системі комп'ютерної алгебри

Зміст ТЧ до дипломної роботи:

Індивідуальне завдання
Анотація
Вступ
1 Огляд предметної області
2 Методи досліджень
3 Опис програмної реалізації СКА з використанням тензорних розрахунків
Висновки
Список літератури

Дата видачі «___» _____ 2019 р. Керівник_____
(підпис)

Завдання отримав _____
(підпис)

Тема: «Тензорні обчислення в системі комп’ютерної алгебри»

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на дипломну роботу.	20.10.2019	
2.	Огляд технічної літератури за темою роботи.	15.11.2019	
3.	Розробка алгоритму	30.12.2019	
4.	Застосування розробленого алгоритму для написання програми	15.02.2020	
5.	Написання теоретичної частини дипломної роботи.	20.04.2020	
6.	Створення слайдів для доповіді та написання доповіді.	25.05.2020	
7.	Аналіз отриманих результатів з керівником та попередній захист курсової роботи.	29.05.2020	
8.	Корегування роботи за результатами попереднього захисту.	03.06.2020	
9.	Остаточне оформлення пояснювальної роботи та слайдів.	09.06.2020	
10.	Захист дипломної роботи	16.06.2020	

Студент _____

Керівник _____

“_____”_____

ЗМІСТ

Анотація

У даній дипломній роботі розглянуто основні галузі використання та типи записів тензорів, огляд бібліотеки Tensorics. Реалізовано опис розробки програми, особливості та специфіка розробки та її призначення. Розглянуто функціонал реалізованої програми з додаванням знімків екрану.

ВСТУП

За переважним типом обчислень системи комп'ютерної математики (СКМ) прийнято поділяти на два класи: інженерні пакети і системи комп'ютерної алгебри. Інженерні пакети створені для ефективного виконання громіздких чисельних розрахунків, а системи комп'ютерної алгебри орієнтовані насамперед на символьні перетворення математичних об'єктів і аналітичне рішення пов'язаних з ними задач. Зауважимо, що в наукових дослідженнях і технічних розрахунках фахівців доводиться набагато більше займатися перетвореннями формул, ніж власне чисельним рахунком, однак, коли з'явилися комп'ютери, то основна увага приділялася автоматизації останніх [1].

Комп'ютерна алгебра виникла в середині XX століття на стику математики та інформатики. Це наука про ефективні алгоритми обчислень математичних об'єктів. Синонімами терміну «комп'ютерна алгебра» є «символьні обчислення», «аналітичні обчислення», «аналітичні перетворення», «формальні обчислення».

Система комп'ютерної алгебри (СКА, англ. Computer algebra system, CAS) - це прикладна програма для символьних обчислень, тобто виконання перетворень і роботи з математичними виразами в аналітичній (символьній) формі.

У системах комп'ютерної алгебри використовуються наступні розділи математики: символьне інтегрування, гіпергеометричне підсумовування, межі, факторизація поліномів, найбільший спільний дільник, метод Гаусса, діофантові рівняння, похідні від елементарних і спеціальних функцій і інше.

Активна розробка систем комп'ютерної алгебри почалася в кінці 60-х років. З тих пір створено значну кількість різних систем, які отримали різну ступінь поширення; деякі системи продовжують розвиватися, інші відмирають, постійно з'являються нові.

Основне призначення систем комп'ютерної алгебри (СКА) - робота з математичними виразами в символьній формі. До базових типів даних СКА відносяться числа та математичні вирази. СКА працюють таким чином [1]:

- математичні об'єкти (алгебраїчні вирази, ряди, рівняння, вектори, матриці та ін.) і вказівки, що з ними робити, задаються користувачем на вхідній мові системи у вигляді символьних виразів;

- інтерпретатор аналізує і переводить символьні вирази до внутрішнього представлення;

- символьний процесор системи виконує необхідні перетворення або обчислення і видає відповідь в математичній нотації.

Алгоритми внутрішніх перетворень мають алгебраїчну природу, що і відображено в назві систем.

Далеко не кожна математична задача має обумовлений існуючими математичними формалізмами аналітичне рішення. Фахівці в областях прикладної і комп'ютерної математики одностайні в думці, що багато практично важливих задач і не можуть бути формалізовані настільки, щоб вирішуватися аналітично, в кращому випадку вони можуть вирішуватися тільки чисельними методами [1].

Що стосується класифікації систем комп'ютерної алгебри, то за критерієм функціонального призначення часто виділяють СКА загального призначення (універсальні) і спеціалізовані. Найбільш відомі системи першої групи: Derive, Mathematica, Maple, Macsyma і її нащадок Maxima, Scratchpad і її нащадок Axiom, Reduce, MuPAD, MathCAD, MATHLAB, Sage, Yacas, Scientific Workplace, Kalamaris. Ці системи дозволяють працювати в широкому діапазоні предметних областей. Системи для вирішення завдань одного або декількох суміжних розділів символьної математики - це спеціалізовані СКА. Прикладами таких систем є: GAP (алгебра груп), Cadabra (тензорна алгебра), KANT (алгебра і теорія чисел) і ін. Тензорні обчислення використовуються в багатьох областях фізики. Слід зауважити,

що у всій своїй могутності формалізм тензорного аналізу проявляється не у всіх областях, досить часто використовують його спрощені варіанти.

Кожна тензорна операція сама по собі досить проста. Однак навіть при стандартних обчисленнях доводиться виконувати безліч елементарних операцій. Ці операції потребують великої уважності і скрупульозності. Саме тому в даній області актуальні різні спрощення нотації, оптимізація операцій.

Одним із завдань систем комп'ютерної алгебри є звільнення дослідника від рутинних операцій, що актуально і в разі тензорного обчислення.

Виникнення спеціалізованої бібліотеки Tensorics по здійсненню складних тензорних обчислень пов’язано з потребами роботи прискорювача елементарних частинок CERN, розміщеного в Швейцарії. Серед інших функціональних можливостей, бібліотека Tensorics надає фреймворк для декларативного опису виразів довільних значень та вирішення цих виразів у різних контекстах. Фреймворк, створений на базі веб-технологій мові програмування Java, забезпечує комфортний спосіб перетворення довільних сигналів з пристроїв до довговічних реактивних потоків. Поєднання цих двох понять забезпечує потужний інструмент для опису модулів з метою онлайн-аналізу [2].

Мета дослідження – побудова системи комп’ютерної алгебри, здатної здійснювати тензорні обчислення великих потоків експериментальних даних.

Предмет дослідження – тензорні обчислення.

Об’єкт дослідження – система комп’ютерної алгебри.

Для здійснення поставленої мети потрібно вирішити наступні завдання:

1. Визначити основні операції тензорного обчислення в рамках бібліотеки Tensorics;

2. Розробити програму трансформації тензорних символьних об’єктів до даних в форматі мови Mathpar;

3. Побудувати систему комп’ютерної алгебри, здатної виробляти математичні операції з тензорними об’єктами

Розділ 1
ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1. 1. Основні галузі використання та типи записи тензорів

Щоб визначити основні види операції з тензорами, розглянемо основні області їх застосування.

1.1.1. Безіндексні обчислення для теоретичних побудов

Безіндексні обчислення зазвичай використовуються в теоретичних побудовах і часто протиставляються компонентним обчисленням.

Подивимося, як можна реалізувати основні тензорні операції в безіндексному випадку [3].

• Додавання тензорів. При складанні двох тензорів валентності

отримуємо тензор валентності :

(1.1)

Додавання тензорів задає структуру абелевої групи.

• Тензорне множення. При тензорному множенні тензора A з валентністю

на тензор D з валентністю отримуємо тензор E з валентністю :

(1.2)

Тензорне множення задає структуру некомутативної напівгрупи.

• Операція згортки. Позначимо операцію згортки тензорів за останніми

індексами через . Тоді під дією цієї операції тензор F з валентністю

переходить до тензору G з валентністю :

(1.3)

• Операція перестановки індексів. Дана операція необхідна для завдання симетрії тензорів (наприклад, комутатора або протикомутатора тензора), для розширення операції згортання до згортки за довільними індексами. Однак в рамках безіндексного підходу позначити цю операцію не можна. Втім, найпростіші симетрії можливо явно вказати в описі об'єкту (при цьому для однозначності доведеться накласти обмеження на валентність).

1.1.2. Векторні обчислення

Векторне числення - найпростіший варіант тензорного обчислення (вектор - тензор валентності один). Вектор a^n розмірності N представляється як сукупність набору компонент , що залежить від базису, і лінійного закону перетворення компонент при зміні базису. Часто використовувані операції - побудови різноманітних диференціальних операторів і заміна базису. Найбільш поширені оператори: градієнт, дивергенція, ротор (специфічна для тривимірного простору).

Для композитних розрахунків необхідно визначити базис, метрику і зв'язність (а відповідно, і коваріантну похідну). У векторному численні широке поширення отримав голономний базис, який будується як сукупність приватних похідних від координат в дотичному розшаруванні і дуального базису як 1-форми в подотичному розшаруванні [3]:

— (1.4)

Можливості підключення і метрика будуються таким чином, щоб коваріантна похідна від метрики дорівнювала нулю:

(1.5)

В цьому випадку зв'язність і метрика узгоджені [3].

Слід зауважити, що в векторних обчисленнях також часто використовується спеціальний неголономний базис, який дозволяє не розрізняти контраваріантні і коваріантні вектори, зберігати розмірність при заміні координат [4]:

— (1.6)

Тут – елемент довжини за відповідною координатою, – коефіцієнти неголономності (в разі ортогональних координат - коефіцієнти Ламі).

1.1.3. Діраковські спінори

Спеціальним випадком тензорних об'єктів є спінор (звані також спін-тензорами). Зокрема спінор є уявленнями групи Лоренца з напівцілою старшою вагою. Звичайні тензори є уявленнями з цілочисельною старшою вагою.

З історичних причин найбільш часто в дослідженнях використовуються діраковські 4-спінори, які застосовують для запису рівнянь Дірака, що описують ферміони зі спіном ½. Діраковські 4-спінори незвідні спінори для випадку $n = 4$ і $s = \pm 2$, де n – розмірність векторного простору $s = n - 2u$ - його сигнатура, u - число від'ємних значень діагонального метричного тензора g_{ab}.

Зазвичай для маніпуляції з діраковськими спінорами використовують γ-матриці, одержувані з рівняння Кліффорда-Дірака [5]:

(1.7)

де - матриці N x N, g_{ab} - метричний тензор, - одинична матриця N x N, N - розмірність спінорного простору:

(1.8)

γ-матриці є елементами алгебри Кліффорда, породжують лінійне перетворення спінового простору.

Оскільки γ-матриці можна розглядати як коефіцієнти переходу від спінового простору до векторного, то більш суворо слід ввести спінові коефіцієнти і записати рівняння (1.7) наступним чином:

(1.9)

Для побудови повної алгебри Кліффорда необхідні ще й добутки - матриць, однак силу (7) досить розглядати лише антисиметризовані добутки:

(1.10)

Також вводиться елемент :

– (1.11)

де e^{abcd} - альтернувальний тензор

Маніпуляції з -матрицями зводяться до набору співвідношень, що прямують з алгебраїчної симетрії, наприклад:

(1.12)

(1.13)

(1.14)

(1.15)

1.1.4. Тензорні обчислення в загальній теорії відносності

Загальна теорія відносності стала першою фізичною теорією, яка вимагала всю міць диференціальної геометрії і тензорних обчислень [6]. В обчисленнях виникають громіздкі тензорні конструкції, які можна спрощувати, враховуючи симетрії тензорів. Зазвичай виділяють одноелементні (monoterm) і багатоелементні (multiterm) симетрії. Одним з основних елементів теорії є тензор Рімана, що володіє як простішими

одноелементними, так і складними багатоелементними симетріями типу тотожностей Бьянкі.

Одноелементні симетрії відповідають простим перестановкам симетрії і задаються групою перестановок. Для тензора Рімана, наприклад, маємо:

(1.16)

Багатоелементні симетрії задаються алгеброю перестановок. Тотожність Бьянкі має вигляд:

(1.17)

Диференційна (друга) тотожність Бьянкі має вигляд:

(1.18)

Симетрії найприродніше задавати за допомогою діаграм Юнга [6]. Причому наявність зумовлених класів тензорів не відміняє необхідність в явному завданні симетрії.

1.1.5. Типи записів тензорів

Таким чином, спираючись на розглянуті вище види тензорних обчислень, можна виділити три типи записів тензорів: компонентний запис, запис з абстрактними індексами і безіндексний запис. Кожен тип має свою специфіку і область застосування.

Компонентні індекси, фактично, перетворюють тензор в набір скалярних величин, використовуваних при конкретних розрахунках. Зазвичай оперувати з компонентними індексами є сенс лише після спрощення тензорного виразу і обліку всіх його симетрій [7].

Безіндексний запис часто використовують, якщо дослідника цікавить не кінцевий результат, а симетрії тензорів. Однак ця форма запису страждає недоліком виразності: тензор розглядається як цілісний об'єкт, відповідно і симетрії можливо розглядати лише ті, які відносяться до тензора в цілому.

Для роботи з об'єктами складної структури доводиться винаходити нові позначення або додавати словесні пояснення. Цю проблему і повинні зняти абстрактні індекси [8].

Абстрактні індекси слід розглядати як удосконалення безіндексного запису тензора. Абстрактний індекс позначає лише приналежність тензора до певного простору, а не проходження тензорного правила перетворення (на відміну від компонентних індексів). В цьому випадку можливий розгляд як симетрії, що охоплюють весь тензор (всі його індекси), так і симетрії окремих груп індексів.

Розділ 2
МЕТОДИ ДОСЛІДЖЕНЬ

2.1. Огляд вимог до проектованої СКА

Розглянемо типову структуру системи комп'ютерної алгебри (СКА).

Внутрішню структуру СКА складають:

- ядро системи;
- інтерфейсна оболонка;
- бібліотеки спеціалізованих програмних модулів і функцій;
- пакети розширення;
- довідкова система.

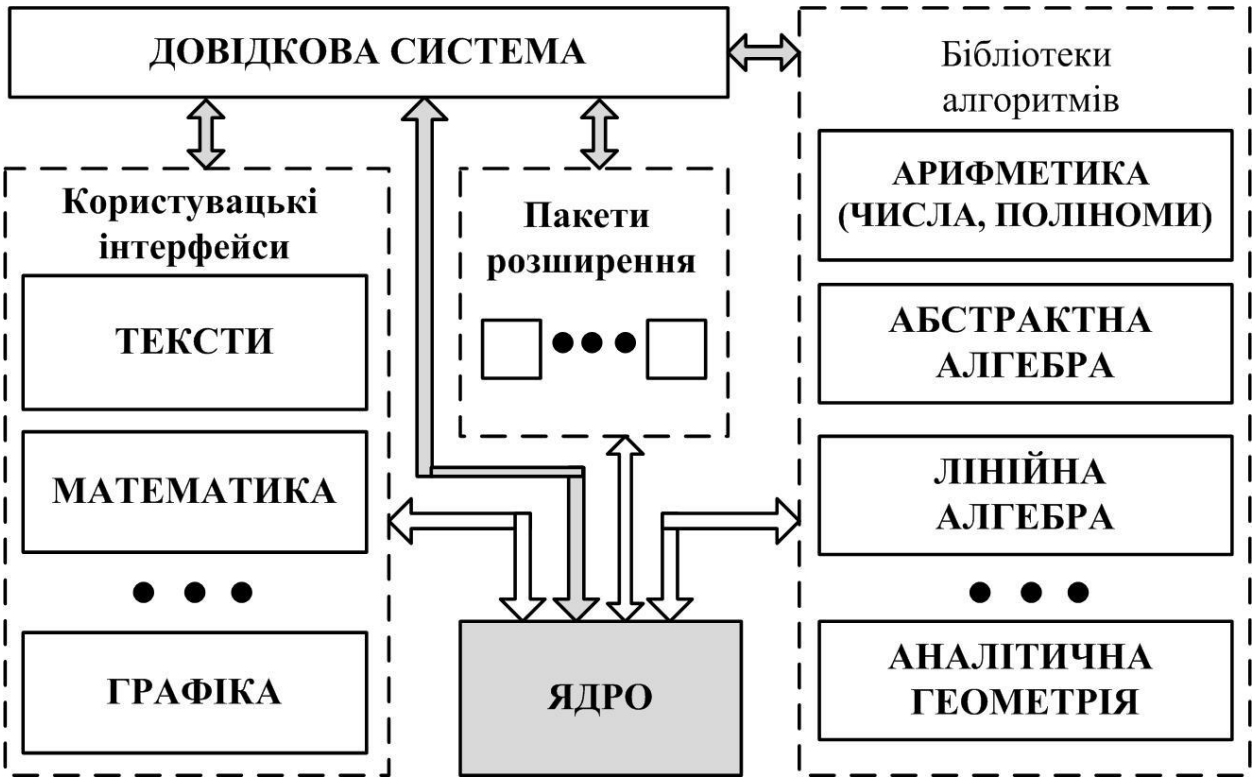

Рис. 2.1. Типова архітектура СКА [9].

Функції ядра, як правило, реалізуються на машинно-орієнтованій мові, тому що потрібна висока продуктивність їх виконання. У деяких СКА оптимізація машинного коду забезпечується, в тому числі, за допомогою часткової реалізації функціональності на мові асемблера або апаратно.

Ядро містить реалізації операторів і вбудованих функцій, що забезпечують виконання аналітичних перетворень математичних виразів на основі системи певних правил.

Обсяг ядра зазвичай обмежують, але до нього додають бібліотеки додаткових процедур і функцій. Розподіл складу підтримуваних системою алгоритмів символьних обчислень між ядром і бібліотеками здійснюється за принципом балансу продуктивності і функціональності з урахуванням поточного стану найбільш поширеного апаратного забезпечення. У більшості комерційних СКА алгоритми обчислень і програмні модулі ядра є ноу-хау розробників і відносяться до розряду ретельно приховуваних даних.

Бібліотеки спеціалізованих програмних модулів і функцій, пакети розширення містять систематизовані за призначенням реалізації алгоритмів обробки абстрактних об'єктів, вирішення типових математичних задач.

Бібліотеки та пакети функціонально розширюють ядро, а також забезпечують можливості програмування алгоритмів не тільки на мові самої системи, а й на мові реалізації, а у багатьох СКА і на основних мовах програмування високого рівня. Інтерфейсні оболонки забезпечують підтримку всіх функцій, необхідних для інформаційних і керуючих взаємодій між системою і користувачами, в тому числі: введення, редагування, збереження, обмін програмами, використання різних апаратних засобів.

У більшості СКА інтерфейсні оболонки різні для різних ОС, при цьому провідні системи комп'ютерної алгебри працюють без перекомпіляції вихідного коду, як на різних апаратних платформах, так і під управлінням різних операційних систем; призначені для користувача інтерфейси забезпечують схожі візуальні сценарії роботи в СКА на різних комп'ютерах, в різних ОС [9].

Довідкова система всіх СКА містить і забезпечує користувачів описами функціональних можливостей і демонстраційними прикладами роботи, інформаційними повідомленнями про поточний стан системи, а також відомостями про математичні основи алгоритмів.

Для СКА є типовою організація і забезпечення діалогу отримання довідок покроково з вкладеними рівнями абстракції та конкретизації інформації. Зазвичай користувачеві доступні: коротка контекстна довідка про функціональне призначення обраного елемента інформація про синтаксис і семантику операторів і функцій мови з пояснювальними прикладами, опис реалізованих варіантів рішення. Інформативність довідкової системи забезпечується обов'язковим описом всіх функцій ядра, інструментами пошуку відомостей про об'єкт СКА по імені, тематичного розділу, ключовими словами. У багатьох системах допомоги містяться навчальні матеріали з розподілом за категоріями користувачів, інтерактивні навчальні курси рішення математичних задач в середовищі системи, деякі навіть мають консультанта-репетитора, який виконує покрокове рішення прикладів з пояснювальними коментарями.

Системи комп'ютерної алгебри дозволяють реалізовувати з використанням комп'ютера аналітичні та чисельні методи розв'язання задач, представляючи результати в математичній нотації, забезпечують графічну візуалізацію, оформлення результатів і підготовку до видання. Використовуючи СКА, можна виконувати в аналітичній формі:

• спрощення виразів до меншого розміру або приведення до стандартного вигляду;

• підстановки символьних і чисельних значень в вирази;

• зміна виду виразів: розкриття творів і ступенів, часткова і повна факторизація (розкладання на множники);

• розкладання на прості дроби, задоволення обмежень, запис тригонометричних функцій через експоненти, перетворення логічних виразів,

• знаходження границь функцій і послідовностей;

• операції з рядами (підсумовування, множення, суперпозиція);

• диференціювання в приватних і повних похідних;

• знаходження невизначених і визначених інтегралів (символьне інтегрування);

• аналіз функцій на неперервність;

• пошук екстремумів функцій і асимптот;

• символьний розв'язок задач оптимізації: знаходження глобальних екстремумів, умовних екстремумів;

• рішення лінійних і нелінійних рівнянь;

• інтегральні перетворення;

• пряме і зворотне швидке перетворення Фур'є;

• інтерполяція, екстраполяція і апроксимація;

• операції з векторами;

• матричні операції: звернення, факторизація, рішення спектральних задач;

• статистичні обчислення;

• машинне доведення теорем.

Якщо завдання має точний аналітичний розв'язок, користувач СКА може отримати це рішення в явному вигляді (зрозуміло, мова йде про завдання, для яких відомий алгоритм побудови рішення).

Більшість СКА забезпечують також:

• числові операції довільної точності;

• цілочислову арифметику для великих чисел;

• обчислення фундаментальних констант з довільною точністю;

• Підтримку функцій теорії чисел;

• Редагування математичних виразів в двовимірної формі (з індексами, звичайними дробами і так далі);

• Побудова графіків аналітично заданих функцій;

• Побудова графіків функцій по табличних значень;

• побудова графіків функцій в двох або трьох вимірах;

• анімацію формованих графіків різних типів;

• використання пакетів розширення спеціального призначення;

• програмування на вбудованій мові.

2.2. Загальні відомості про бібліотеку Tensorics

Tensorics - це Java фреймворк для обробки багатовимірних даних. Основним об'єктом називається тензор, який являє собою сукупність значень довільних типів, що виражені координатами в N-мірному просторі (в деякому сенсі як колекція з асоційованими ключами) [2].

```
import org.tensorics.core.lang.Tensorics.*;
Tensor<Double> degrees = builder(City.class, Day.class)
  .put(at(SAN_FRANCISCO, YESTERDAY), 11.5)
  .put(at(SAN_FRANCISCO, TODAY), 12.5)
  .put(at(NEW_YORK, YESTERDAY), 2.5)
  .put(at(NEW_YORK, TODAY), 3.5)
  .build();
Double todayInNewYork = degrees.get(NEW_YORK, TODAY); // 3.5
```

Кожний раз, коли треба шукати одну або декілька з наведених нижче функцій, може стати в нагоді Tensorics:

- багатовимірні масиви;
- асоціативні масиви;
- математичні операції над багатовимірними даними.

Основні характеристики

Незважаючи на те, що в бібліотеках основна увага приділяється багатовимірній обробці даних, а її назва походить від "тензор" (багатовимірний об'єкт в математиці), він надає кілька додаткових концепцій, які доповнюють одне одного. Функції розроблені для того, щоб плавно працювати разом, але кожну з них, звичайно, можна використовувати і окремо. Найважливішими з них є:

- Тензори довільної розмірності як центральний об'єкт;
- У тензорів можуть бути елементи будь-якого (Java) типу;

- Структурні та числові операції на тензорах.

- Внутрішній DSL Java (вільний API) для всіх операцій над скалярами та тензорами;

- Кількість (значення - пара одиниць);

- Повна підтримка тензорного числення;

- Поширення помилок та валідності для величин та тензорних величин;

- Сценарій всіх функціональних можливостей із відкладеним виконанням, що відкриває можливості для паралельної обробки та масового розподілу обчислень;

- Утиліти для створення "ланцюжків рішень" у Java, які імітують концепцію, подібну до відповідності випадків у масштабі.

Тензори. Назва Tensorics походить від тензор, який у математиці являє собою багатовимірну структуру даних, записи якої адресовані за індексами. Основна особливість тензора від Tensorics полягає в тому, що розмірність ідентифікується типом java (класом). Екземпляри відповідного типу позначаємо як координати. Потім значення тензора адресується у просторі N-мірних координат набором об'єктів (екземплярами класів координат) [2].

У тензора від Tensorics є параметр одного типу, тип значень, який він містить, зазвичай позначається як <V>. Тому структура даних тензора може використовуватися як контейнер для будь-якого типу Java. Однак деякі операції над тензорами будуть можливі лише для певних типів значень (наприклад, математичних операцій).

Оскільки поняття Tensorics та синтаксис найкраще пояснюються під час практичного ознайомлення, то буде використовуватися наступний приклад. Розглянемо аналіз погоди: Набір даних складається з даних про погоду з різних міст та часів. Клас Місто і Час визначені, а деякі константи - екземплярами. Наприклад:

```
City SF = City.ofName("San Francisco");
City LA = City.ofName("Los Angeles");
Time T1 = Time.of("2017-01-01 15:00");
```

```
Time T2 = Time.of("2017-01-02 15:00");
Tensor<Double> degrees;
/* створення пропущено */
```

Доступ до величин. Припускаючи наведені вище константи, ми можемо просто отримати від тензора значення температури [2]:

```
Double t = degrees.get(T1, SF);
```

Як видно тут, це виглядає дуже схоже на отримання значень з карти, з такими важливими відмінностями:

- Метод get тензора приймає N аргументів, по одному для кожного виміру;
- Метод отримання тензора ніколи не повертається до нуля. Це призведе до відповідного винятку, якщо в тензорі немає значення для заданого набору координат.

Загалом слід зазначити, що всі методи в бібліотеці розроблені для швидкого виходу з ладу. Це особливо важливо, оскільки \ tensorics {} завдяки своєму гнучкому API не може покладатися на перевірку часу компіляції у багатьох випадках, і тому деякі помилки з’являються лише під час виконання.

Набір N координат називається положенням у Tensorics. Таким чином, код з наведеного списку еквівалентний

```
Position position = Position.of(T1, SF);
Double t = degrees.get(position);
```

Основна точка входу. Інтерфейси об'єктів Tensorics зберігаються дуже тендітно і зазвичай забезпечують лише абсолютно необхідні методи. Усі інші операції над цими об'єктами базуються на статичних методах, що діють на них. Основною точкою введення для цих методів (що містить усі методи, не характерні для певних типів значень) є клас \ код {Tensorics}. Цей клас містить також, наприклад, метод делегування методу \ code {Position.of ()}:

```
Position position = Tensorics.at(T1, SF);
/* with static import: */
Position position = at(T1, SF);
```

Використання статичного імпорту для цього дозволяє стислий код, який буде особливо важливим при створенні тензорів [2].

У всіх наступних прикладах коду припускається, що коли є звичайний виклик методу, то це статичний метод із класу Tensorics (або іншими словами, що Tensorics імпортується статично).

Створення тензорів. Усі наявні в даний час реалізації тензорів незмінні. Звичайний спосіб їх створення - через builder. Наприклад, щоб створити наш тензор температури і ввести в нього 4 значення, нам доведеться зробити щось на кшталт:

```
Tensor<Double> degrees =
   builder(City.class, Time.class)
       .put(at(SF, T1), 12.5)
       .put(at(SF, T2), 14.2)
       .put(at(LA, T1), 17.5)
       .put(at(LA, T2), 19.2)
       .build();
```

Знову ж, синтаксис дуже схожий на побудову незмінної карти. Дійсно, це ще один спосіб тензору \ tensorics {}: як карта від позиції до значення, і вона може бути перетворена в одне:

```
Map<Position, Double> degreesMap = mapFrom(degrees);
```

Скаляр/ Тензор може мати нульові розміри. Цей конкретний тензор позначається як \ scalar у \ tensorics {}. У позиції Position.empty () воно має дорівнювати одному значення. Скаляр можна просто створити за допомогою статичного заводського методу

```
Scalar<Double> scalar = scalarOf(2.5);
```

Форма. Досі ми просто використовували тензор як своєрідну карту з комбінованими ключами. Однак реальні властивості виявляються лише тоді, коли мова йде про трансформації. Як передумову цього важливо ввести таке поняття серед тензорів Shape: так само, як карта має свій набір ключів, тензор у Tensorics має форму. Вона переважно описує структуру тензора, без

його значень. Переважно вона містить таку інформацію: розміри тензора (наприклад, Time.class та City.class у наведеному вище прикладі) та наявні позиції в тензорі.

Форму можна отримати з тензора та використати для нашого прикладу, наприклад [2]:

```
Shape shape = degrees.shape();
Set<Class<?>> dims = shape.dimensionSet();
/* Contains Time.class and City.class */
int dim = shape.dimensionality();
/* Will be 2 */
Set<Position> poss = shape.positionSet();
/* contains the 4 positions */
int size = shape.size();
/* Will be 4 */
```

Структурні операції. Витягування субтензорів. Однією дуже поширеною структурною операцією є витяг субензорів з тензора:

```
Tensor<Double> sfDegrees = from(degrees).extract(SF);
```

Це призведе до одновимірного тензора, який містить лише координати типу Time. Додаткова операція до цього називається тензорами злиття.

Примітка: хоча в методі get кількість координат завжди повинна точно відповідати розмірності тензора (інакше метод викине), метод \ code {extra} приймає будь-який підмножина розмірів як аргумент; метод \ code {get} повертає значення тензора, тоді як метод \ code {extra} повертає знову тензор. Це означає, що якщо координати для всіх розмірів подаються як аргументи методу вилучення, то повертається нульовий розмірний тензор. Повернений тензор може бути порожнім у випадку, якщо на витягнутих координатах відсутні елементи.

Математичні операції. Однією з важливих мотивацій використання тензорів є, звичайно, наявність простих та інтуїтивних способів виконання математичних операцій над ними. Хоча структурні операції - як описано дотепер - можуть виконуватися на тензорах будь-яких типів значень, зрозуміло, що математичні операції можна виконувати лише з тензорними значеннями певних типів.

Математичні структури. Tensorics не суворо обмежує типи, над якими можна виконувати математичні операції, але забезпечує механізм розширення, за допомогою якого - в принципі - можна додати математичні можливості для будь-якого типу значень. На практиці це має сенс (і є лише необхідним) для обмеженої кількості типів цінностей. Механізм розширення вимагає забезпечити (тензорні значення $ a, b, c $) [2]:

Дві двійкові операції, додавання (+) та множення (*) із наступними властивостями:

обидва, + і * є асоціативними: $ a + (b + c) = (a + b) + c $; $ a * (b * c) = (a * b) * c $.

і +, і * мають елемент ідентичності (називається "0" для +, "1" для *): $ a + 0 = a $; $ a * 1 = a $.

і +, і * мають зворотний елемент (називається '-a' для +, '1 / a' для *): $ a + (-a) = 0 $; $ a * 1 / a = 1 $.

обидва, + і * є комутативними: $ a + b = b + a $; $ a * b = b * a $.

* є розподільним по +: $ a * (b + c) = a * b + a * c $.

Математично кажучи, дві операції утворюють алгебраїчну структуру поля \ cite {wikipedia-field} над значеннями тензора <V>: --- * Дві додаткові бінарні операції: Power ($ a ^ b $) та Root ($ \ sqrt [b] {a} $). * Функція перетворення значень тензора до і з подвоєних. --- Якщо ці операції надаються загальним класам підтримки \ tensorics {}, всі маніпуляції, що базуються на наведеному нижче, будуть доступні шляхом успадкування від цих класів підтримки. Найбільша перевага підходу, який використовується в Tensorics для визначення поля (і використання зовнішніх методів для

обчислень - не методів елементів поля), полягає в тому, що він (технічно) не накладає жодних обмежень на тип значення і, таким чином, уникає, наприклад, обертайте об'єкти за потребою в польових реалізаціях інших математичних бібліотек (наприклад, Apache Commons Math \ cite {apache-commons-math}).

Незважаючи на те, Tensorics на даний момент забезпечує реалізацію цих вимог для парних. Для спрощення цих дуже часто потрібних операцій він також забезпечує клас зручності (\ код {TensoricsDoubles}) зі статичними методами делегування класам підтримки. Така зручність буде недоступною для коробки для спеціальних типів значень, але їх можна легко додати аналогічним чином. Щоразу, коли в наступних прикладах є виклик трейлінг-методу, будемо вважати, що це статичний метод з класу \ code {TensoricDoubles}.

Одинарні операції. Поряд з операціями на тензорах, класи підтримки також надають зручні операції для ітераторів. Наприклад [2]:

```
Iterable<Double> v = Arrays.asList(1.0, 2.0);
Iterable<Double> negv = negativeOf(v);
Double vsize = sizeOf(v);
Tensor<Double> t; /* creation omitted */
Tensor<Double> negt = negativeOf(t);
Double tsize = sizeOf(t);
```

Основна статистика. Деякі дуже прості статистичні методи представлені поза рамками. Для ітерабелів результати є просто типовими для елементів ітерабельного:

```
Iterable<Double> v = Arrays.asList(1.0, 2.0);
Double avg = averageOf(v);
Double sum = sumOf(v);
Double rms = rmsOf(v);
```

З іншого боку, для тензорів застосування статистичних операцій зазвичай виконується лише в одному вимірі. Це відповідає зменшенню тензора на

один вимір. Наданий текучий API відображає це (продовжуючи наш приклад раніше):

```
/* All these return Tensor<Double>: */
reduce(degrees).byAveragingOver(Time.class);
reduce(degrees).byRmsOver(Time.class);
reduce(degrees).bySummingOver(Time.class);
```

Бінарні операції. Розрахунок операцій між двома тензорами, нарешті, дає найбільшу користь. Усі ці операції починаються з використання методу \ code {TensoricDoubles.calculate (…)}:

```
/* degrees and offset are Tensor<Double> */
calculate(degrees).plus(offset);
calculate(degrees).minus(offset);
calculate(degrees).elementTimes(other);
calculate(degrees).elementDividedBy(other);
/* All these return Tensor<Double> */
```

Тут обидва, лівий і правий операнди вважаються тензорами. Однак голі значення також підтримуються з обох сторін і будуть неявно перетворені в скаляри. Чотири вищезазначені операції є найпростішими, оскільки вони засновані на операціях з елементами: Кожен елемент у лівому тензорі вимагає лише відповідного елемента у правому тензорі для отримання відповідного елемента в результуючому тензорі. Однак для цього потрібні деякі інші міркування: що станеться, якщо обидва операнди мають різні форми? Цю проблему можна вирішити у два етапи, які в \ tensorics {} називаються \ emph {мовлення} та \ emph {переформатування}. Вони пояснюються в наступних двох розділах. \ Tensorics {} має дуже модульний спосіб поводження з такими випадками: користувачі можуть використовувати (і навіть реалізовувати) різні стратегії в особливих випадках. Якщо нічого не вказано, буде застосовано чутливий за замовчуванням [2].

Переформатування/ Це простіше з двох можливих невідповідностей форми: Це означає, що обидва розглянутих тензора мають однакові розміри, але вони мають значення для різних позицій (наприклад, один має менше записів, ніж інший). Поведінка за замовчуванням для цього випадку полягає в тому, що результуючий тензор матиме лише значення для позицій, які містяться в кожному з тензорів (Перетин набору позицій).

Трансляція. Термін \ emph {трансляція} запозичений з бібліотеки python \ emph {numpy} \ cite {numpy-github}. Незважаючи на те, що базовий принцип дуже схожий на нумерологічний, існує кілька істотних відмінностей, які випливають з того, що numpy використовує багатовимірні масиви з цілими індексами, тоді як Tensorics визначає його розміри за класами: Стратегія мовлення за замовчуванням у \ tensorics {} транслює всі виміри, які є \ emph {not}, доступними в одному тензорі, у формі другого тензора. Іншими словами, до іншого тензора буде доданий вимір, який не присутній в одному, і всі значення координат відповідного виміру потенційно будуть поєднані з усіма положеннями іншого тензора. Наприклад [2]:

```
Tensor<Double> temps =
  builder(Time.class)
    .put(at(T1), 10.5)
    .put(at(T2), 12.2)
    .build();
Tensor<Double> offsets =
  builder(City.class)
    .put(at(SF), 2.0)
    .put(at(LA), 7.0)
    .build();
Tensor<Double> result = calculate(temps).elementTimes(factors);
/* Will contain 4 positions: (SF, T1), (SF, T2), (LA, T1), (LA, T2) */
```

Результат буде точно таким же тензором, що і побудований в \ lstref {buildingATensor}. Виконуючи двійкові операції, обидва операнди спочатку

транслюються, а потім перетворюються. Це гарантує правильність розмірів, а потім всі відповідні елементи працюють на відповідних партнерах.

Внутрішній продукт. Це саме особливе множення двох тензорів є в основному узагальненням матричного множення. Синтаксис настільки ж простий, як це може бути:

обчислити (градуси), час (інше);

Щоб цей результат був очікуваним, слід розрізняти коефіцієнти ко- та контра-варіанту. У \ tensorics {} це розрізнення досягається за допомогою наступного механізму: За замовчуванням координати вважаються противаріантними. Коваріантні координати змушені успадковувати з класу \ code {Коваріант <C>}, де загальний параметр \ код {<C>} є типом відповідної противаріантної координати.

2.2.1. Фреймворк Streamingpool

Серед інших функціональних можливостей, бібліотека Tensorics надає фреймворк для декларативного опису виразів довільних значень та вирішення цих виразів у різних контекстах. Фреймворк Streamingpool забезпечує комфортний спосіб перетворення довільних сигналів з пристроїв до довговічних реактивних потоків. Поєднання цих двох понять забезпечує потужний інструмент для опису модулів з метою онлайн-аналізу [10].

Пристрої прискорювачів передають свої вимірювальні дані через асинхронні канали, на які можуть підписатися додатки вищого рівня. Ці дані можна розглядати як потоки елементів даних. Нещодавня еволюція технологій надає концепції, поверх яких створений фреймворк Streamingpool для управління такими потоками і використовується в операційних додатках, як описано в [1]. Приєднання до одного потоку на протязі тривалого часу недостатньо, але коли декілька потоків об'єднані, і певне рішення, засноване на певній логіці, може бути знайдено. Повністю базувати таку логіку на асинхронних операторах, як це надає RxJava (технологія, яку використовує і

надає Streamingpool), виявляється надмірно складною і важкою для читання та налагодження. Додатково, досвід попередніх розробок додатків у CERN (наприклад, Система програмного блокування програмного забезпечення [2]) показав, що такий аналіз сприятливо базувати на технології моментальних знімків.

Коли розпочала роботу програма програмного забезпечення BE-OP-LHC над новою системою діагностики ін'єкцій LHC (Large Hadron Collider), саме ці проблеми виникли. Метою було вміти формулювати умови таким чином, щоб їх могли читати та розуміти непрограмісти, і в той же час забезпечити необхідний комфорт (наприклад, підтримка IDE та заповнення коду) для людей, які мають сформулювати такі умови. Крім того, оскільки вже досягнуто хороших результатів при аналогічному підході до розробки тестового аналізу в LHC [3,4], було вирішено орієнтуватися на внутрішню специфічну мову домену Java (DSL).

Це нарешті призвело до структури аналізу, описаної в цьому розділі. Він побудований модульним способом і складається з наступних компонентів, які можна використовувати окремо або разом, залежно від сфери використання:

• Streamingpool [1] забезпечує абстрагування потоків даних, що надходять від прискорювальних пристроїв, і в контексті рамки аналізу використовується для всіх видів асинхронної обробки, необхідних для створення знімка, який потім передається справжньому каналу. (наприклад, запуск, буферизація та відображення потоків).

• DSL аналіз базується на розширенні до DSL, наданому бібліотекою Tensorics [3]. Він описує логіку, яка застосовується до даних, і може, наприклад, використовуватись окремо, наприклад, для аналізу статичних даних (наприклад, шляхом витягування з служби реєстрації даних LHC).

Streamingpool - це структура з відкритим кодом [1,5], яка резюмує спосіб виявлення, створення та управління ними реактивних потоків.

У Streamingpool кожний потік однозначно ідентифікується завдяки Streamid. Streamid забезпечує абстрагування того, що розробник хоче отримати як потік (наприклад, потік даних з апаратного пристрою). Враховуючи Streamid, можна виявити пов'язаний потік (повернутий як Pubiisher <T>) за допомогою DiscoveryService. Цей сервіс запитує Streamingpool для потоку, який ідентифікується наданим StreamId. Якщо потік вже був виявлений до Streamingpool, то повертає того ж Publisher <T>, інакше запускає його ліниве (lazy) створення, а потім кешує його для подальших запитів.

Аналіз DSL. Логіка онлайн-аналізу описується розширенням модуля AnalysisModule. AnalysisModule забезпечує особливий DSL на основі Java для вираження логіки аналізу. Ця функція дає можливість розробнику реалізовувати складну логіку, маючи повну гнучкість мови програмування Java. Це також робить перевірений тип аналізу, оскільки можна перевірити на помилки під час компіляції, і він дає повні можливості автоматичного завершення під час розробки.

Твердження. Основними складовими для аналізу є твердження. Вони використовуються для визначення умов для перевірки під час аналізу. Як приклад, розглянемо, що користувач хоче стверджувати, що "виробляються протони", але цю умову слід враховувати лише тоді, коли "запитуються протони" (оскільки лише тоді умова має сенс).

Це можна сформулювати за допомогою аналізу DSL, як показано в Лістингу 1.

Listing 1: Затвердження в прикладі коду модуля аналізу.

```
whenTrue ( PROTONS_ARE_REQUESTED )
. thenAssertBoolean ( PROTONS_ARE_PRODUCED )
. isTrue ();
```

Тут обидві константи (PROTONS_ARE_REQUESTED і PROTONS_ARE_PRODUCED) відносяться до StreamIdBasedExpressions, як буде пояснено далі.

Приклад показує, що твердження загалом складається з двох булевих виразів: умова та передумова. Попередня умова формулюється в пункті WhenTrue (...) ("протони запитуються" у вищевказаному прикладі). Умова, що представляє реальну логіку бізнесу, визначається тоді, використовуючи тодішнє сімейство методів. Якщо передумови пропущено, вони вважатимуться завжди правдивими

Виходячи з цієї визначеної логіки, кожне твердження може повторно перевірятися на результат в одному з наступних станів, коли здійснюється аналіз:

• SUCCESSFUL: якщо булева умова в результаті істинна

• FAILURE: коли стан має результат хибного.

• NON_APPLICABLE: якщо вираз попередньої умови було оцінено як хибне (замасковане).

• ERROR: коли сталася очікувана помилка під час оцінки висловлення умови або передумови.

Після оцінки кожен модуль аналізу також дає загальний результат, який представляє резюме всіх результатів тверджень, що містяться в модулі. Можливі значення результату:

• SUCCESSFUL: всі твердження в модулі є SUCCESSFUL або NON_APPLICABLE (маскуються).

• FAILURE: принаймні одне твердження дало значення ERROR або FAILURE, тому загальний аналіз вважається збоєм.

• ERROR: під час оцінювання AnalysisExpression сталася несподівана помилка.

Зазвичай Інтернет-аналіз має потоки як вхідні дані (ідентифіковані потоками, як описано раніше). Надається спеціалізований модуль для аналізу – StreamBasedAnalysisModule для подальшого розширення для висвітлення

цих випадків. Цей клас забезпечує додаткову функціональність DSL для роботи з потоками даних. Точна поведінка модуля онлайн-аналізу залежить від підтипу StreamBasedAnalysisModule, від якого він успадковується. Наразі доступні три типи поведінки:

• Логіка неперервного AnalysisModule оцінюється кожного разу, коли будь-який вхідний потік аналізу отримує нові дані, аналіз перераховується з останнім значенням для всіх інших потоків. Подальша параметризація поведінки не можлива і не потрібна.

Логіка з TriggeredAnalysisModule оцінюється кожний час, коли певний потік випромінює елемент. Автор модуля може вказати цей потік як частину DSL цього типу модуля аналізу. За ступенем надходження даних вхідних потоків зберігається лише останнє значення на потік. Коли потік тригера дає значення, аналіз оцінюється, використовуючи останні значення для вхідних потоків, які були раніше збережені.

Лістинг 2: Синтаксис для параметризації тригерного аналізу.

```
new TriggeredAnalysisModule () {
{triggered ().by( aTriggerStreamId );} });
```

Нарешті, BufferedAnalysisModule задає аналізи, які діють на буферні вхідні потоки, визначені відповідно до стратегії BufferEvaluation. У цьому випадку DSL пропонує специфічні методи для визначення часу, коли буферизація повинна починатися та закінчуватися. Ключовий клас для буферизації в Інтернеті.

Аналіз - це BufferedStreamExpression, що є вузлом на графіку аналізу, який приймає потік як параметр. Під час оцінювання модуля аналізу BufferedStreamExpression видає буфер, що містить значення, збережені з потоку даних, ідентифікованого зазначеним Streamid. У такий спосіб розв'язують буферизацію та визначення потоку (streamid). Приклад

фрагменту DSL для визначення потоків початку та закінчення буфера виглядає приблизно так:

Лістинг 3: Приклад буферизованого модуля аналізу.

```
new BufferedAnalysisModule () {{
buffered ()
. startedBy ( startStreamId )
. endedOnEvery ( endStreamId )
.or ()
. endedAfter ( Duration . ofSeconds (10));}});
```

Частина тензорних виразів фреймворкового аналізу базується на тензорних виразах, які надаються проектом Tensorics з відкритим кодом [6]. Цю частина бібліотеки надає Java DSL для опису операцій над вхідними даними, які згодом можуть бути застосовані до вхідних даних у різних контекстах. Технічно кажучи, вираз Tensorics - це вузол у прямому ациклічному графіку (як правило, дерево у простих випадках, як, наприклад, проілюстровано на рис. 1), який може мати дітей і може бути вирішений до певного значення. Основними перевагами цього підходу, порівняно з безпосередньо виконанням Java-коду, є наступне:

• Діаграма є незмінним об'єктом Java і може бути серіалізована (з деякими обмеженнями). Це дає можливість, наприклад, правила аналізу можуть бути відправлені під час виконання до іншого процесу Java (можливо, на інший хост). Це дозволяє розподілити обчислення без перерозподілу.

Інформація, що міститься у діаграмі, може бути легко доступною для подальшої обробки. Ця функція використовується в рамках аналізу, наприклад автоматично будувати значущі імена для тверджень. Далі це дозволить створити загальні компоненти GUI, які можуть, наприклад, відобразити детальну інформацію про причини, чому певні твердження не вдалися. Це аналогічний підхід, як у [3].

Оскільки повна інформація графіка доступна, вона є тривіальною, щоб визначити необхідні вхідні дані (залишити вузли), що дуже важко, якщо буде використано звичайне виконання Java. Це ключова концепція для описаної структури аналізу: виділений streamFactory визначає необхідні вхідні потоки з діаграми та передає їх як вхід до механізму роздільної здатності.

Сам механізм роздільної здатності можна оптимізувати без зміни (або навіть перекомпіляції) логіки діаграми. Це буде описано в наступному розділі.

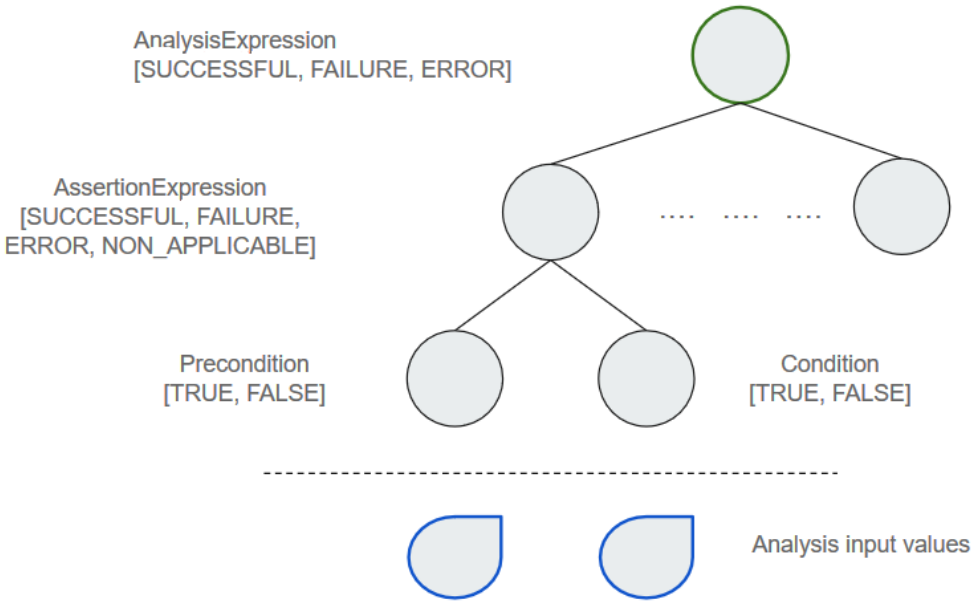

Рис.2.2. Структура дерева для онлайн-аналізу.

Перш за все слід зазначити, що, хоча механізм вираження походить від бібліотеки Tensorics, ми тут не використовуємо жодних тензорів. Насправді механізм вираження є загальним, так що він може в принципі поширюватися на будь-яку необхідну операцію.

Розглянемо обчислення середньої інтенсивності пучка із загальної інтенсивності пучка та кількості пучків. Відповідна логіка виглядала б приблизно так:

Лістинг 4: Приклад тензорного виразу.

Expression <Double > totalIntensity ;

```
Expression <Double > numberOfBunches ;
/* Creation omitted */
Expression <Double > avgBunchIntensity =
TensoricDoubleExpressions
. calculate ( totalIntensity )
. dividedBy ( numberOfBunches );
```

Після виконання цього фрагмента коду змінна avgBunchintensity являє собою одне дерево з однією дією (поділом) та двома операндами у вигляді листя (загальна інтенсивність та кількість пучків).

Роздільна здатність фактичного значення такого виразу делегована ResolvingEngine. Це в найпростішому випадку виглядатиме так:

Лістинг 5: Розв’язання виразу.

```
ResolvingEngine engine ;
/* Creation omitted */
Double result = engine
. resolve ( avgBunchIntensity );
```

Двигун, що вирішує, - це об'єкт, який здатний пройти по діаграмі та вирішити вузли. Це відбувається завдяки використанню так званих резолюцій, кожна з яких здатна вирішити певний тип виразу за своїм значенням. Детальний алгоритм точно такий же, як описано в [4], і тому тут не повторюється. Однак, порівняно з DSL, використовуваним у системі відстеження тестових прискорювачів (AccTesting) [3,4], мова виразу тензорів має такі вдосконалення:

• Дозволений тип може бути будь-якого типу Java, тоді як для мови AccTesting потрібні спеціальні типи обгортки.

• Бібліотека Tensorics забезпечує підтримку декількох кроків нашого поля (наприклад, обчислення на основі скалярів, тензорів), які дуже важко

реалізувати в програмі DSL AccTesting через обмежені типи даних, які використовуються там.

Будь-який вузол графіка може нести значення (називається ResolExpression) або бути невирішеним (AbstractDeferredExpression). В останньому випадку значення вузла обчислюється одним резонатором тензорів, коли графік розв’язаний. Концепція Resolvers забезпечує необхідну розв'язку виразу та відповідне виконання коду: Наприклад, якщо вузол реалізує оператор суми, користувацький Resolver повинен запитувати результат операндів, а потім повертати суму значень. У наведеному вище прикладі змінна загальна напруженість може, наприклад, присвоюється певному виразу, що представляє змінну в системі реєстрації даних та момент часу. Відповідний роздільник буде витягувати відповідне значення з системи реєстрації даних, коли запитується механізмом вирішення.

Розв’язувальний контекст. Під час вирішення кінцевого результату графів вузли вирішуються знизу вгору (спочатку залишаються) і всі підсумкові результати збираються до об'єкту, який називається ResolvingContext, таким чином, щоб вони були доступні для розділювачів батьківського вузла. Роздільний механізм забезпечує додаткове переосмислення методу resolve(..), який дозволяє подавати в початковому контексті. Контекст можна розглядати як просту карту від виразу до вирішеного значення, а також може бути побудований як такий. Якщо всі входи надані, то вирішальний двигун може здійснювати обчислення лише за допомогою своїх внутрішніх резолюцій (без будь-якого спеціального - наприклад, для запиту бази даних журналів). У цьому випадку наведений вище приклад зводиться до простого розрахунку:

Еволюція програмних технологій

Лістинг 6: Приклад тензорного виразу.

```
Expression <Double > t =
Placeholder . ofName ("t");
Expression <Double > n =
```

```
Placeholder . ofName ("n");;
Expression <Double > avg =
TensoricDoubleExpressions
. calculate (t). dividedBy (n);
ResolvingContext ctx =
ResolvingContext .of(t, 2.2 e11 , n, 2.0);
ResolvingEngine engine =
ResolvingEngines . defaultEngine ();
Double result = engine . resolve (avg , ctx);
/* results in 1.1 e11 */
```

Розширення до Streamingpool. Фреймворковий аналіз використовує саме функцію попередньо заповненого контексту - як описано в попередньому розділі - для того, щоб з'єднати вирази Streamingpool та Tensorics. Необхідні розширення на потоковий пул інкапсульовані в проекті потокового streamingpool [7]. Основні частини цього розширення:

• StreamidBasedExpression <T> може використовуватися як вузол в DSL і несе в якості корисного навантаження відповідний Streami d <T>.

• AnalysisStreamId має в якості корисного дерева дерево виразів, яке описує логіку аналізу. Її листя потенційно можуть бути екземплярами StreamIdBasedExpression <T>

• Кілька різних реалізацій StreamFactory відповідають за визначення необхідних вхідних потоків з дерева виразів AnalysisStreamId, шукаючи відповідні потоки, застосовуючи необхідну стратегію буферизації, правильно попередньо заповнивши ResolvingContext і, нарешті, запитувати ResolvingEngine для остаточного результату аналізу.

Іншими словами, попередньо заповнений ResolvingContext являє собою знімок вхідних даних, що підлягають аналізу. Такий підхід робить справжню логіку синхронною та відтворюваною. Розширення, передбачене на майбутнє, - це також можливість записувати вхідні дані та пізніше

відтворювати лише знімок або навіть окремі потоки. Це дозволить налагодити логічні проблеми або навіть невідповідності асинхронної поведінки.

Фреймоворковий аналіз базується на новітніх технологіях, доступних у світі Java. На додаток до вищезазначених Tensorics Expressions і Streamingpool, Spring [8] використовується для підключення залежностей, а реактивні потоки [9] використовуються для обробки входів (використовуючи RxJava [10] як реалізацію Java).

Код фреймоворкового онлайн-аналізу широко використовує новітні можливості Java: Методи за замовчуванням в інтерфейсах Java застосовуються для складання класів утиліт, щоб зробити тестування одиниць якомога простішим. Крім того, публічний API фреймворку був розроблений таким чином, щоб бути сумісним з лямбдами Java 8, де це можливо, з метою оптимізації досвіду кодування користувача.

Фреймоворк, представлений в роботі над LHC, використовує Spring для введення залежності. Для полегшення прийняття системи онлайн-аналізу було створено набір проекту Spring Boot ([5]). З огляду на те, як працює Spring Boot, лише додавши проект для запуску як залежність у додаток Spring Boot, розробник матиме попередньо налаштоване середовище для аналізу в Інтернеті, готове до використання. Налаштування все ще можливі через переосмислення значень за замовчуванням, необхідних для будь-якого конкретного сценарію.

Основними реалізаціями реактивних потоків для Java є Project Reactor [3] та RxJava [4]. Обидві реалізації поділяють одне і те ж поняття щодо поводження з помилками: підписка на джерело руйнується, якщо виникає помилка, і абонент повинен повторно підписатися (підключитися). Ця стратегія є запобігання помилки згортання потоку. Ця стратегія корисна в таких сценаріях, коли створення підписок дороге, має побічні ефекти або коли користувач хоче створити постійний аналіз даних у режимі реального часу, не будучи одночасно кодом обробки помилок та повторною підпискою

на потоки. Для подолання проблеми обробки помилок, кожен потік ^ у Streamingpool пропонує паралельний потік Throwable, тому, що всі помилки, які згорнули б реактивний потік, відхиляються до потоку помилок. Таким чином, напр. Виняток із тайм-аута не збільшить підписку, і користувач обробляє виняток відповідно (наприклад, звітуючи про GUI). Такий підхід відкриває двері до тривалих Р реактивних потоків, де виникають проблеми з виробництвом, переробкою або передача одного елемента не заважає здатності потоку.

Бібліотека Streamingpool має просту, але потужну api-бібліотеку. Основна його мета - забезпечити абстракцію над управління реактивними потоками в програмному застосуванні.

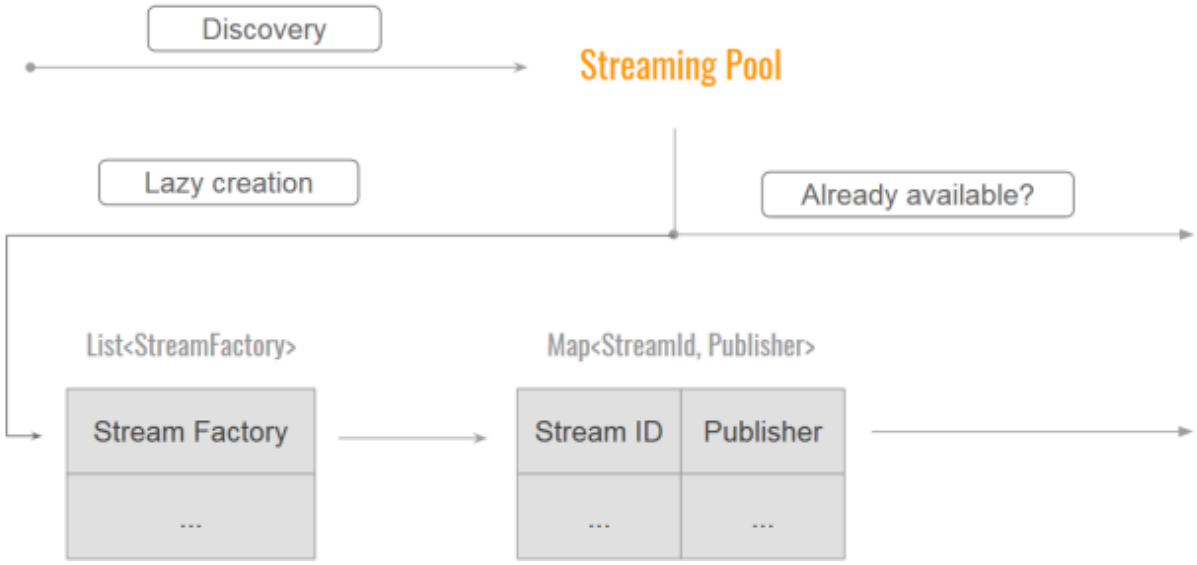

Рис. 2.3. Потік архітектури Streamingpool. StreamId

Streamingpool був розроблений головним чином за новою програмою діагностики ін'єкцій ЛНС [6]. У цьому додатку пуловий потік поєднується з бібліотекою тензориків [7,8], щоб отримати рамку аналізу багаторазового використання [9]. Хоча в даний час це найскладніший випадок використання, Streamingpool також використовується для різних простих програм керування, наприклад, той, що відображає час, що залишився для м'якого пуску LHC для інжекційних кікерів або графічних інтерфейсів користувача,

які керують хроматичністю [10] та з'єднанням [11] LHC. Також у випадках, коли не використовуються GUI, почали використовувати потоковий пул. Наприклад, розділ CERNs TE-MPE-MS надає (використовуючи Streamingpool) потоки з декодованими пучками пучка, які можуть реєструватися та використовуватися іншими програмами.

З перших днів Streamingpool було зрозуміло, що наступним логічним кроком у розвитку має стати транспортування потоків по мережі. На той час технологія реактивних потоків була ще досить молодою, тому було вирішено відкласти вибір технології для цього і зосередитись на функціональності, описаному у вищезазначених розділах. Тим часом технологія розвивалася і доступно кілька варіантів. Наприклад, проект Spring включив реактивні контролери у свою версію 5.0. Використання gRPC [12] як мережевого рівня - це ще одна можливість.

Другий аспект, на якому буде зосереджено подальший розвиток, - це в тому числі більше діагностики та налагодження функцій. Завдяки стандартизованому підходу в Streamingpool можуть бути побудовані загальні компоненти (наприклад, графічні інтерфейси користувача), які, наприклад, може показувати відносини між потоками або часовою структурою пов'язаних елементів. Одним із прикладів такого загального компонента графічного інтерфейсу, який вже існує, є панель JavaFx, яка показує винятки з усіх потоків помилок, що надаються пулом, які можуть бути включені до будь-якої програми, використовуючи Streamingpool як резервний (рис. 2.4.).

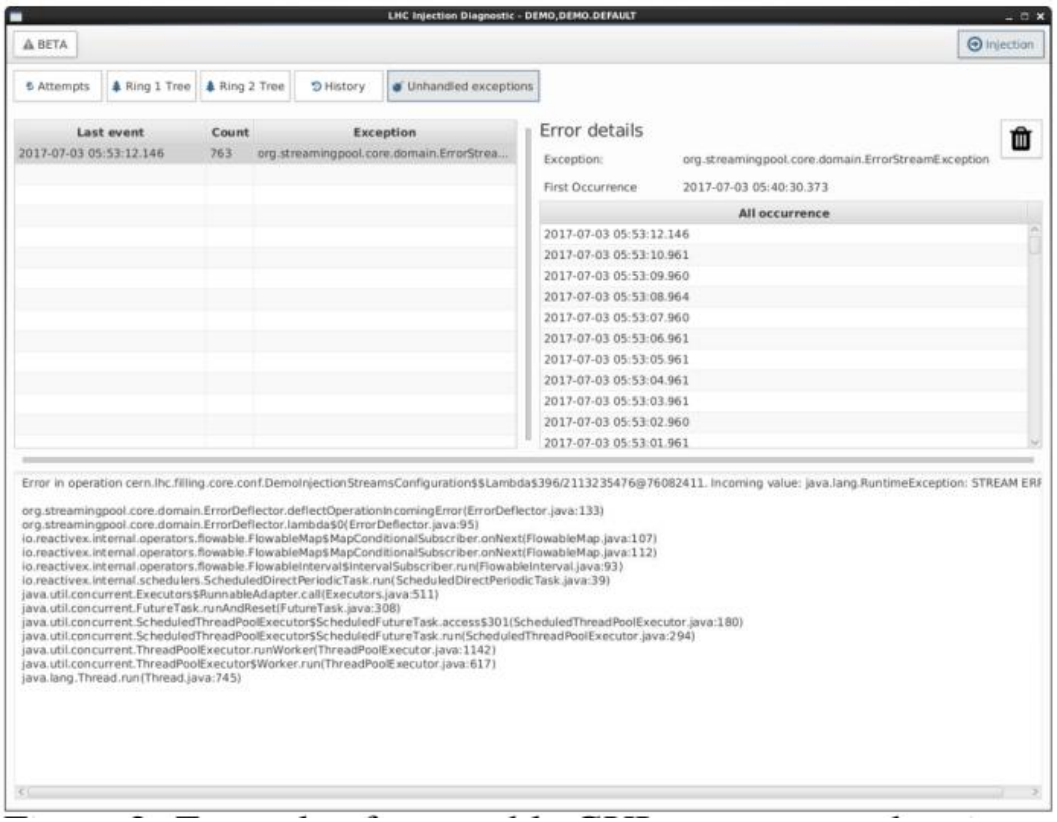

Рис. 2.4. Приклад багаторазового використання GUI компонента, що показує відхилені помилки програми.

2.3. Практичні методи реалізації концепції реактивного програмування

2.3.1. Концепція реактивного програмування

Реактивне програмування - це парадигма асинхронного програмування, пов'язана з потоками даних і поширенням змін. Це означає, що стає можливим легко висловлювати статичні (наприклад, масиви) або динамічні (наприклад, випромінювачі подій) потоки даних через використовувану мову програмування.

Реактивна система - архітектурний паттерн, який задовільняє деякий набору правил (reactive manifesto). Даний моніфест був розроблений в 2013 році для усунення невизначеності. Справа в тому, що на той момент в Європі і США термін «reactive» був занадто надмірним. Кожен розумів по-своєму, яку систему можна назвати реактивної. Це породжувало величезну плутанину, і в підсумку був створений маніфест, який встановлює чіткі критерії реактивної системи.

Подивимося на картинку з маніфесту і розберемо більш детально, що означає кожен пункт:

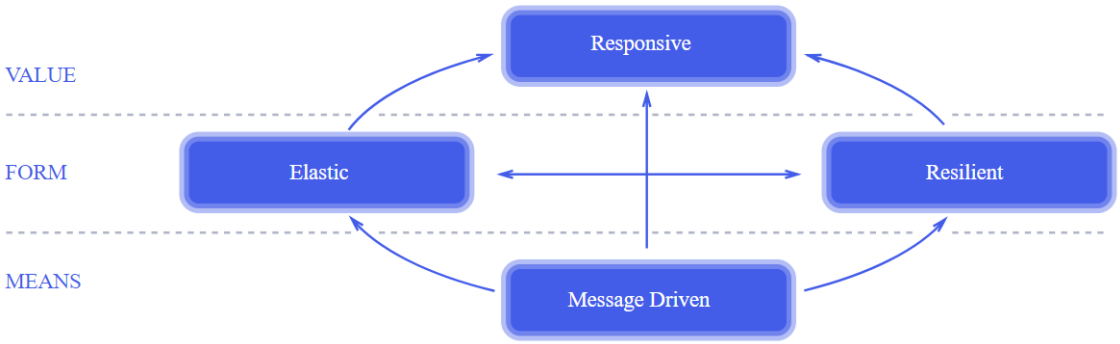

Рис. 2.5. Схематичний вираз концепції реактивного програмування.

Responsive. Даний принцип говорить нам про те, що розроблювана система повинна відповідати швидко і за певний заздалегідь заданий час.

Крім того система повинна бути досить гнучкою для самодіагностики і лагодження.

Що це означає на практиці? Традиційно при запиті деякого сервісу ми йдемо до бази даних, виймаємо необхідний обсяг інформації і віддаємо її користувачеві. Тут все добре, якщо наша система досить швидка і база даних не дуже велика. Але що, якщо час формування відповіді набагато більше очікуваного? Крім того, у користувача міг пропасти інтернет на кілька мілісекунд. Тоді всі зусилля по вибірці даних і формування відповіді пропадають. Згадайте gmail або facebook. Коли у вас поганий інтернет, ви не отримуєте помилку, а просто чекаєте результат більше звичайного. Крім того, цей пункт говорить нам про те, що відповіді і запити повинні бути впорядковані і послідовні.

Resilient. Система залишається в робочому стані навіть, якщо один з компонентів відмовив. Іншими словами, компоненти нашої системи повинні бути достатньо гнучкими і ізольованими один від одного. Досягається це шляхом реплікації. Якщо, наприклад, одна репліка PostgreSQL відмовила, необхідно зробити так, щоб завжди була доступна інша. Крім того, наш додаток повинен працювати в безлічі екземплярів.

Elastic. Даний принцип говорить про те, що система повинна займати оптимальну кількість ресурсів в кожен проміжок часу. Якщо у нас високе навантаження, то необхідно збільшити кількість екзепляри додатки. У разі малої навантаження ресурси вільних машин повинні бути очищені. Типовий інструменти реалізації даного принципу: Kubernetes.

Message Driven. Тут починається найважливіший пункт для Java-розробника. Спілкування між сервісами має відбуватися через асинхронні повідомлення. Це означає, що кожен елемент системи запитує інформацію з іншого елемента, але не очікує отримання результату відразу ж. Замість цього він продовжує виконувати свої завдання. Це дозволяє збільшити користь від системних ресурсів і більш гнучко керувати помилками, які

виникають. Зазвичай такий результат досягається через реактивне програмування.

Розділ 3
ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СКА З ВИКОНАННЯМ ТЕНЗОРНИХ РОЗРАХУНКІВ

Було реалізовано програму(бібліотеку), яка дозволяє виконувати тензорні обчислення. Даною програмою доповнено сервіс MathPartner (його встановлено на сервері Києво-Могилянської академії mathpartner.ukma.edu.ua) – хмарної системи символьних обчислень – і тих можливостей, які вона надає для вдосконалення освітнього процесу у вищій школі[13]. Хмарна математика є універсальним інструментом застосування математичних знань, що сприяє розвитку прикладної математики[13].

Сервіс MathPartner з'явився в 2011 році і був одним із перших серед хмарних систем [14]. Сервіс використовує мову Mathpar. Текст у мові Mathpar, на відміну від операторів, потрібно брати в подвійні лапки. Оператори розділяються крапкою з комою або текстом. Всі оператори виконуються послідовно, і результат останнього оператора з'являється після тексту. Якщо ж зустрічаються команди друку, то виводиться не останній оператор, а результати цих команд [13]. В основі мови Mathpar лежить широко використовувана математиками і фізиками мова TeX, яку зазвичай використовують для набору математичних текстів. Mathpar є процедурною мовою програмування. Використовуються процедури і функції [13]. Для більш детального ознайомлення з основними конструкціями мови Mathpar можна скористатися літературою [15; 16].

Правило запису тензорів у сервісі MathPartner

Тензор створюється за допомогою такого синтаксису:

{}_{...}^{...}SYMBOLIC_{...}^{...}, де

- *_{...} – лівий/правий коваріантний індекс;*
- *^{...} – лівий/правий контраваріантий індекс;*
- *SYMBOLIC – символ тензора;*

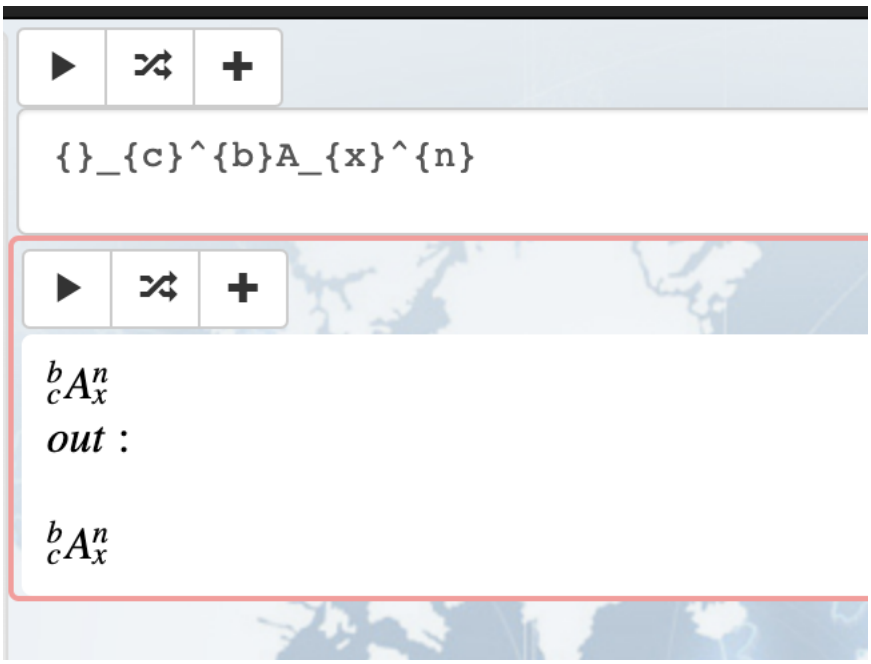

Рис. 3.1. Запис та парсинг тензора.

Основні обчислювальні операції з тензорами, реалізовані в бібліотеці

- Додавання/Віднімання. Операція складання може бути застосована тільки до тензорів, які мають однакову кількість нижніх і верхніх індексів (тобто до тензорів одного і того ж рангу і типу). Якщо нам дано два тензора одного і того ж рангу і типу, то, алгебраїчно підсумовуючи кожен компонент першого тензора і відповідний компонент другого, ми, очевидно, отримаємо тензор того ж рангу і типу, що і складові. Зазначена операція називається складанням, а отриманий результуючий тензор називається сумою(різницею) двох тензорів.

{}_{c}^{b}A_{x}^{n} + {}_{c}^{b}B_{x}^{n} (3.1)

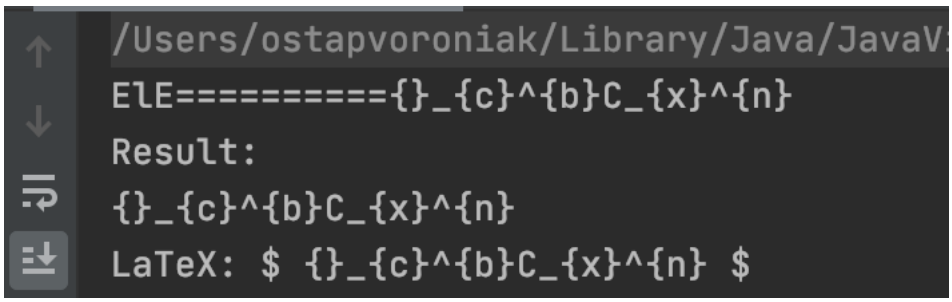

Рис. 3.2. Результат додавання двох тензорів з формули (3.1).

{}_{c}^{b}A_{x}^{n} - {}_{c}^{b}B_{x}^{n} - {}_{c}^{b}C_{x}^{n} (3.2)

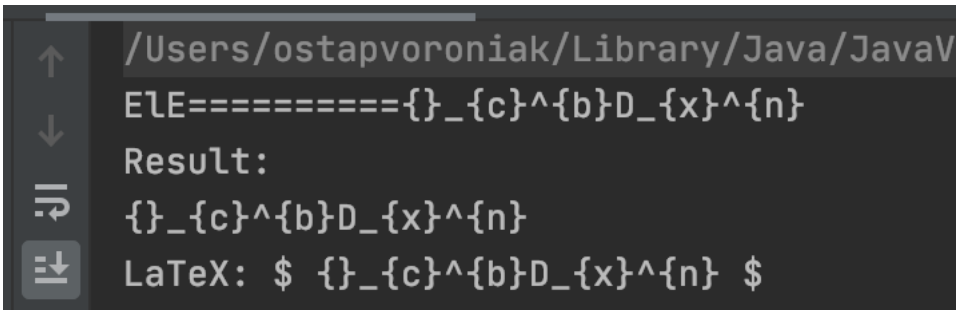

Рис. 3.3. Результат віднімання трьох тензорів з формули (3.2).

Коли користувач захоче додати два тензори різних рангів, то програма логічно виведе помилку, що ці тензори не мають однакову кількість нижніх і верхніх індексів.

{}_{c}^{b}A_{xaaaaa}^{n} + {}_{c}^{b}B_{x}^{n} (3.3)

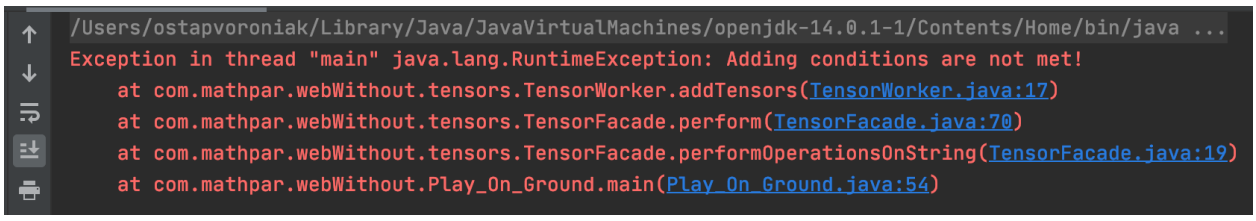

Рис. 3.4. Результат додавання тензорів з різними індексами з формули (3.3).

- Множення. Визначимо добуток двох тензорів будь-якого рангу і типу. Перемноживши кожний компонент першого тензора на кожен компонент другого тензора, отримаємо тензор, ранг якого дорівнює сумі рангів двох тензорів. Зазначена операція називається множенням, а отриманий результуючий тензор - добутком двох тензорів.

  B_{x}^{saa} * C_{x}^{a} (3.4)

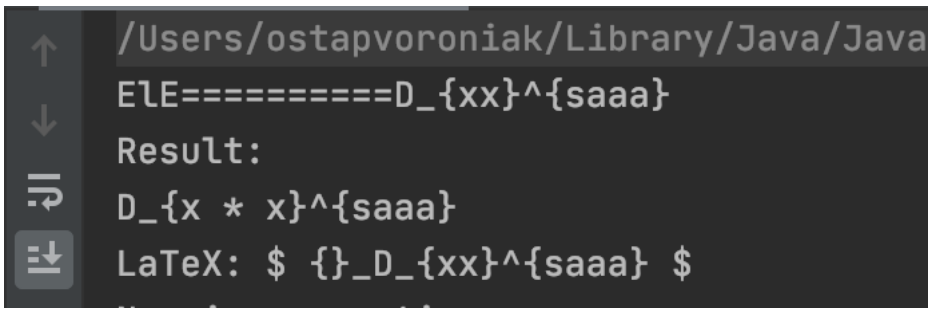

Рис. 3.5. Результат множення двох тензорів з формули (3.4).

- Згортання (скорочення індексів). Операція згортання може бути застосована тільки до змішаних тензорів;

{}_{c}^{b}A_{xqq}^{nqq} (3.5)

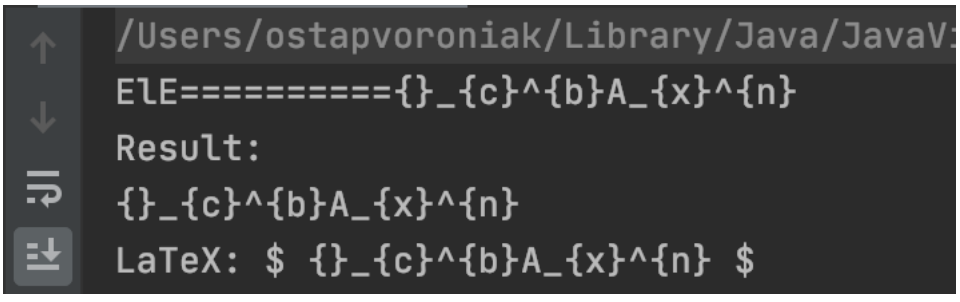

Рис. 3.6. Результат згортки тензора з формули (3.5).

Програма також надає можливість працювати з комбінованими операціями:

A_{xx}^{saaa} + B_{x}^{saa} * C_{x}^{a} - D_{xx}^{saaa} (3.6)

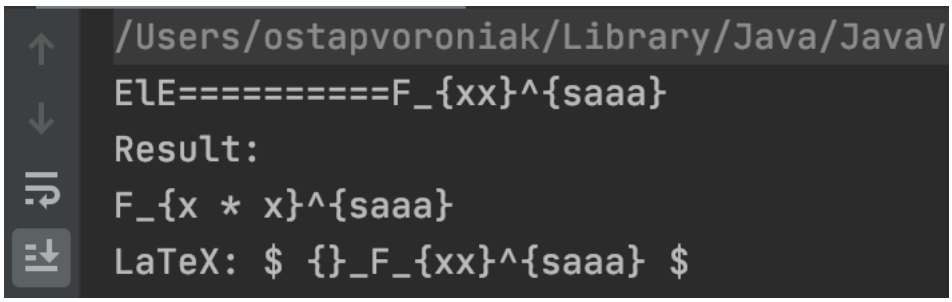

Рис. 3.7. Результат згортки тензора з формули (3.6).

Коли користувач введе неправильний запис тензору, то програма виведе відповідну помилку.

{}_{c}^{b}A_{xqqqq}^{nqq}}}} (3.7)

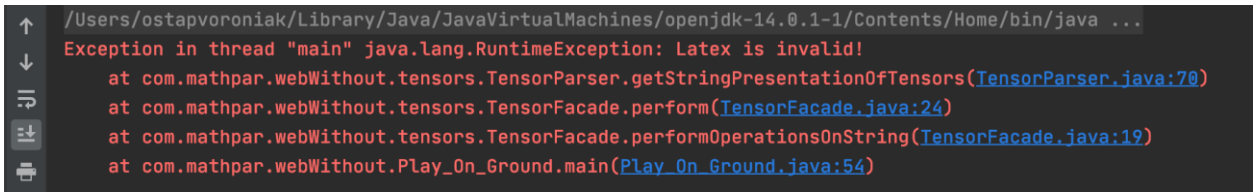

Рис. 3.8. Результат введення неправильних даних з формули (3.7).

Висновки

Отже, **метою** мого дослідження була побудова системи комп'ютерної алгебри, здатної здійснювати тензорні обчислення.

Я познайомився з існуючими системами комп'ютерної алгебри, поняттям тензора, основними галузями використання та обчислювальними операціями тензорів. Розробив програмний продукт, за допомогою якого можна виконувати основні обчислювальні операції з тензорами. Ознайомився з проектом MathPartner та за допомогою парсера та створеного програмного продукту доповнив його тензорними обчисленнями.

В цій роботі досліджено:

- призначення програми;
- особливості та специфіку розробки;
- огляд функціоналу створеної програми;

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Клименко В.П., Ляхов А.Л., Гвоздик Д.Н. Современные особенности развития систем компьютерной алгебры// Математичні машини і системи, - 2011, - № 2 – С. 2-16.

2. https://tensorics.org/

3. Королькова А.В., Кулябов Д.С., Севастьянов Л.А. Тензорные расчеты в системах компьютерной алгебры// ПРОГРАММИРОВАНИЕ, - 2013, - № 3, - С. 47-57.

4. Кулябов Д.С., Немчанинова Н.А. Уравнения Максвелла в криволинейных координатах//Вестник РУДН. Серия «Математика. Информатика. Физика». - 2011. - № 2. - С. 172-179.

5. Сарданашвили Г.А. Современные методы теории поля. Т. 2. Геометрия и классическая механика. М.: УРСС, 1998.

6. Гердт В.П., Тарасов О.В., Ширков Д.В. Аналитические вычисления на ЭВМ в приложении к физике и математике / / Успехи физических наук. 1980. Т. 130. С. 113-147. http: / / ufn.ru/ra/articles /1980/ 1/d/

7. Барут А., Рончка Р. Теория представлений групп и ее приложения. М.: Мир, 1980.

8. Penrose R., Rindler W. Spinors and Space-Time. Two-Spinor Calculus and Relativistic Fields. Cam-gridge University Press. 1987. V. 1. 472 p.

9. Таранчук В. Б. Основные функции систем компьютерной алгебры. — Минск: БГУ, 2013. http://elib.bsu.by/handle/123456789/46210

10. A. Calia, K. Fuchsberger, M. Gabriel, M.-A. Galilée, G.-H. Hemelsoet, M. Hostettler, M. Hruska, D. Jacquet, J. Makai, ”Streaming Pool - Managing Long-Living Reactive Streams for Java”, ICALEPCS2017, Barcelona, Spain (2017),

11. L. Ponce, J. Wenninger, J. Wozniak, ”Operational Experience with the LHC Software Interlock System”, ICALEPCS2013, San Francisco, CA, USA (2013), MOPPC069

12. M. Audrain et al., ”Using a Java Embedded DSL for LHC Test Analysis”, ICALEPCS2013, San Francisco, CA, USA (2013), THPPC079

13. Малашонок Г.І. Хмарна математика MathPartner у Києво-Могилянській академії. – 2017. – Режим доступу: http://ekmair.ukma.edu.ua/handle/123456789/12537

14. Malaschonok G. I. Way to Parallel Symbolic Computations [Електронний ресурс] / G. I. Malaschonok // Облачные вычисления. Образование. Исследования. Разработка: [материалы конференции]. – Москва, 2011. – Режим доступу: www.unicluster.ru/conf/2011/docs/TSU.MalaschonokG.I. pdf. – Назва з екрана.

15. Малашонок Г. И. О проекте параллельной компьютерной алгебры / Г. И. Малашонок // Вестник Тамбовского университета. Серия: Естественные и технические науки. – Тамбов, 2009. – Т. 14, вып. 4. – С. 744–748.

16. Малашонок Г. И. Система компьютерной алгебры MathPartner / Г. И. Малашонок // Программирование. – 2017. – № 2. – С. 63–71