**УДК 004.413**

**Д. Зербіно, Ю. Цимбал**
Національний університет “Львівська політехніка”,
кафедра автоматизованих систем управління

# ТЕНДЕНЦІЇ НЕВИЗНАЧЕНОСТІ У МОВАХ ПРОГРАМУВАННЯ



**Розглянуто підходи до створення мов програмування, які призначені для розв’язання інтелектуальних задач. Введено різні типи невизначеностей та проаналізовано їхню роль в підвищенні рівня інтелектуальності та зручності мови програмування.**

**In the article the approaches to the creation of programming languages that are intended for solving the intellectual problems have been studied. Different types of uncertainity have been introduced and their role in increasing the intellectual level and the convenience of programming languages has been presented.**

## Вступ

Реальне життя характерне тим, що необхідно розв’язувати велику кількість неточно поставлених задач. У таких задачах припускається певна невизначеність вхідних даних. Невизначеність може бути як у постановці задачі, так у і засобах її вирішення. Під невизначеністю даних в програмуванні будемо розуміти можливість застосування до них апарату теорії нечітких множин. Під невизначеністю алгоритмів – можливість уточнення або доповнення алгоритму під час роботи програми, а також послідовного вибору його альтернативних варіантів (з можливістю повернення стану програми до точки вибору). Основна мета введення невизначеності в програмуванні – це спрощення розуміння програми. Нечіткість мови програмування не означатиме, що вона не містить чітких операцій. Просто чітка операція розглядається як дія з ймовірністю 100%, а нечітку дію під час виконання програми в разі необхідності можна замінити іншою. Для програмування невизначеність дає певну зручність, тому що програміст, коли не знає як саме скласти певну програму, може ввести в неї нечіткі поняття. В процесі програмування ці поняття будуть уточнюватись і структуризуватись. Отже, невизначеність стає частиною власне технології програмування.

## Історія проблеми

Тривалий час думка про можливість розроблення мов програмування з використанням невизначеностей була джерелом різного роду жартів. Наприклад, висувалася “ідея” створення мови C+– (“більш або менш C” – за аналогією з мовою C++). Зокрема, пропонувалося перевантажити операції “>” та “<” як “краще за” та “гірше за”, а “>>” та “<<” як “значно краще за” та “забудьте про це !” [1]. Тобто, зверталась увага на недостатньо очевидні особливості синтаксису мови.

Із розвитком нечіткої логіки та теорії нечітких множин на початку 1990-х років розпочалася розробка мов програмування на їх основі. Серед таких мов можна виділити fuzzy programming language (fpl) фірми togai infralogic [2] та dcu fuzzy compiler, розроблений в університеті м. Дубліна (ірландія) [3]. Мови нечіткого програмування надають можливість оголосити та ініціалізувати вхідні змінні, описати відповідні нечіткі множини (або функції належності) для їхніх значень і утворити базу нечітких знань та ввести правила їх опрацювання. Як ми бачимо, цей напрямок тісно пов’язаний з класичною теорією баз знань і менше стосується власне програмування.

Серед новітніх підходів до розвитку мов програмування зазначимо підхід, запропонований засновником нечіткої логіки л. заде [4]. Зокрема, запропоновано при створенні мови максимально наблизити її складові (зокрема значення змінних та операції над ними) до конструкцій, вживаних у природній мові людини. наприклад,

*“якщо (X є “дуже малим”) – то (Y є “не дуже великим”),*

або

*якщо (X є "великим") – то (Y є "малим").*

тоді можна поставити запитання на зразок:

*"яким є значення X, якщо Y є "великим"* ?

Для опису таких задач уведено концепцію узагальненої мови обмежень (generalized constraint language, gcl). Мови, які можна розробити на основі цієї концепції, потенційно нададуть більше інтелектуальності у представленні задач та їх подальшому вирішенні.

Усі ці підходи у той чи інший спосіб спрямовані на представлення фактів, хоча і в нечіткій формі. Але, на наш погляд, для програмування важливим є вибір альтернативи у програмі. нечітка альтернатива надає можливість програмі самостійно вибрати шлях виконання (послідовність наступних операцій). Залежно від результатів обчислень за певною альтернативою програма зможе повернутись до вибору іншої альтернативи. Тобто, вказаний вище приклад можна було б переформулювати так:

*якщо (X є "великим") – то (Y є "малим" або Z є "середнім").*

Отже, невизначеність даних доповнюється певною невизначеністю коду.

**Типи невизначеностей**

Зі сказаного вище можна зробити висновок, що мови програмування з невизначеностями певною мірою сприятимуть розвитку теорії самоорганізації алгоритмів. Можна сказати і навпаки, що алгоритми, які самостійно організуються, стимулюватимуть розроблення таких мов програмування. Раніше розв'язати неточно поставлені задачі було неможливо, тому що будь-яка класична алгоритмічна мова працює з конкретною інформацією. Тепер з'являється певний рівень розуміння (як у розробників систем програмування, так і прикладних програмістів), який дає змогу розв'язувати неточно поставлені задачі.

будемо розрізняти такі типи невизначеностей.

- Перший тип невизначеності – це класичне розуміння нечіткої логіки [5], тобто, якщо ймовірність більша за задану межу, то обирається одне рішення (або певне значення), якщо менша – то інше. Якщо ймовірність певного значення дуже мала, то це не означатиме, що воно не буде братися до уваги. Просто насамперед для знаходження розв'язку розглядатимуть ймовірніші варіанти.
- Другий тип – це невизначеність самої мови програмування, тобто, один і той самий текст програми можна інтерпретувати по-різному. Наприклад, один і той самий оператор "*приєднати*" можна трактувати по-різному залежно від контексту, до якого належить цей оператор. Відповідно до свого рівня інтелектуальності система програмування може правильно інтерпретувати цей текст чи неправильно (з погляду програміста).
- Третій тип невизначеності полягає у тому, що невідомо, коли програма повинна припинити пошук результату, тобто сама прийняти рішення про те, що цей напрямок перебору варіантів є безперспективним для пошуку і зробити так званий "бектрекінг".

**Зв'язок невизначеностей з концептуальністю**

Концептуальність потрібна для того, щоб розібратись в "океані" невизначеностей, виділити структуру задачі, відокремити головні дані від другорядних [6]. Концептуальність необхідно розглядати одночасно з невизначеністю ще і тому, що вона зв'язана з різноманітністю інтерпретації понять, що введені у програмах, або з інтерпретацією самого тексту програми. Невизначеність також стосується і структури задачі, і того, які дані вважати головними, а які – другорядними.

Непоганим прикладом концептуальності в програмуванні є мова логічного програмування prolog, де спочатку формулюються факти, задаються правила отримання нових даних, далі дається постановка задачі, формулюється мета. Невизначеним залишається лише рішення задачі, яке система знаходить сама. деякі автори намагалися ввести нечіткість в логічне програмування [2, 3], але поширеними їхні системи не стали.

Програма в такій системі складається з набору правил: якщо деякий об'єкт має деякі властивості, то до нього можуть бути застосовані певні дії, причому кожна дія має певну ймовірність. Наприклад, дія номер 1 може бути застосована з ймовірністю p1, дія номер 2 – з ймовірністю p2. Все, що залишилося: p3 = 1-(p1+p2) – ймовірність того, що до об'єкта не може бути застосована жодна дія.

Концептуальні мови програмування необхідні в такій сфері, де не справляються звичайні алгоритмічні мови програмування – в області нечітко поставлених інтелектуальних задач.

**Підвищення інтелектуальності систем**

Один з основних параметрів інтелектуальних систем – міра інтелектуальності – визначається можливістю роботи з нечіткою інформацією. Інтелектуальність системи полягає у тому, що вона сама доповнює та інтерпретує ту інформацію, котра вважається нечіткою чи неповною, тобто, система на свій розсуд додає іншу інформацію, спираючись на власний досвід. Коефіцієнт якості інтелектуальності такої системи залежить від того, наскільки точно вона вгадала те, чого хоче користувач. Є суттєва різниця в тому, чи система сама вирішує, які треба брати дані для обчислень, чи вона запрограмована на вибір певних даних. системи першого типу, як правило, інтелектуальні, другого – ні.

З невизначеністю близько зв'язані деякі допущення. для виконання певних дій система повинна обумовлювати, що: 1) існують деякі дані; 2) існують певні рішення на основі цих даних; 3) для рішення необхідно зробити певні припущення. Тоді невизначеність усувається тим, що система, зробивши такі припущення, перевіряє кожен з варіантів рішення, пропонованих системою, і обирає один з них.

Отже, коли система сама вибирає дані, ми не знаємо заздалегідь, які саме дані вона вибере. У цьому полягає невизначеність і інтелектуальність системи.

**Нечіткі операції**

Однією з особливостей нечіткого програмування є можливість перенесення властивостей одних об'єктів на інші. Це може бути використано при пошуку рішень за аналогією. взагалі операція нечіткого пошуку об'єктів є складовою частиною нечіткого програмування. В результаті нечіткої операції можна отримати нечіткий результат, якому можна надати нечітку оцінку. Тобто, влаштовує нас даний результат чи ні можна сказати лише після певного порівняння з деякими іншими результатами. У структурі такої мови програмування можна виділити фільтри ситуацій та аналізатори різного рівня і типу: аналізатор обмежень, аналізатор близькості мети, аналізатор можливих дій і т.д. з кожним фільтром (аналізатором) пов'язаний еквівалентний пояснювальний текст (природною мовою). Фрагменти цього тексту можуть бути використані системою при виявленні помилок та їх роз'ясненні.

Розглянемо конкретний приклад – запис числа в масив. в алгоритмічних мовах це абсолютно чітка операція – якась комірка масиву отримує значення. В мовах програмування з невизначеностями деякий масив пов'язується з певним поняттям, що розбите на градації – комірки масиву. Тоді операція присвоєння виглядає як припущення, що деяка градація поняття отримує певне (неточне) значення, зміст якого буде зрозумілим в подальшому тексті програми. Також довільне присвоєння необхідно розглядати як наділення певного об'єкта певними властивостями. Однак, на відміну від класичних мов програмування, це присвоєння не є остаточним, і система може повернутися до цієї операції для вибору іншої альтернативи або для уточнення значення, яке було присвоєне.

Для нечіткого програмування дуже важливим є формулювання критеріїв – що саме вважати правильною програмою, а також – коли можна знайти рішення і коли їх шукати безперспективно. Дуже важливим є формулювання теоретичних можливостей системи, а також їх класифікація. Нечітке програмування дає змогу, по-перше, представити думку людини у вигляді програми для комп'ютера, тобто максимально їх зблизити; а по-друге, систематизувати власну логіку, або представити своє бачення певної проблеми.

**Виправлення помилок у нечітких програмах**

Всі поняття в мові програмування з невизначеностями створює сам програміст і далі з ними оперує. Отже, з появою невизначеності помилки в програмах стають нечіткими, тобто неможливо точно сказати, чи є в програмі помилки. тобто, виправлення помилок в таких мовах програмування полягає у поступовому збільшенні визначеності. При складанні нечітких програм можна використовувати позитивні та негативні приклади результатів, тоді як у чітких програмах використовуються лише позитивні приклади.

У системі можливі виправлення помилок лише в тому випадку, коли вона розвивається. тому під час програмування виникають дві серйозні пов'язані задачі – нарощування абстрактного понятійного рівня алгоритму, за яким працює програма, і виправлення помилок, які найчастіше пов'язані з неправильним трактуванням цих понять. Чим більший рівень абстракції програми, тим більшою є її невизначеність і тим більше понять вона охоплює. Певний рівень невизначеності в мові програмування якраз полегшить нарощування понятійного рівня та виправлення помилок, пов'язаних з неправильним трактуванням понять, які були введені в мові програмування.

Отже, програміст і програма діють як одне ціле – аналізуючи позитивні та негативні приклади, поступово уточнюють та структуризують ті поняття, які введені в програмі.

**Висновки**

Свого часу більшість програмістів вибрали алгоритмічні мови, відмовившись від використання декларативних мов програмування. Значною мірою це було викликано нечіткістю методів складання декларативних програм. Тому ми намагалися обґрунтувати необхідність розроблення мов програмування з врахуванням невизначеності, притаманної мисленню людини, а також зробили деякі припущення щодо розвитку таких мов. Звичайно, що практична реалізація таких мов програмування – це справа майбутнього. сьогодні необхідно зацікавити широкі маси програмістів "таємничою" невизначеністю і зробити її зручним інструментом під час розроблення програм.

*1. Пратт Т., Зелковиц М. Языки программирования: разработка и реализация. – СПб.: Питер, 2002. – 688 с. 2. Fuzzy Programming Language: http://www.ortech-engr.com/fuzzy/fpl.html. 3. Yan J., Ryan M., Power J. Using Fuzzy Logic: Towards Intelligent Systems. – Prentice-Hall, 1995. 4. Zadeh L.A. Toward a Perception-based Theory of Probabilistic Reasoning // Melo-Pinto P., Teodorescu H.-N., Fukuda T. (eds.) Systematic Organisation of Information in Fuzzy Systems. – IOS Press, 2003. – P. 3–5. 5. Yager R..R., Zadeh L.A. Fuzzy Sets, Neural Networks and Soft Computing. – New York, Van Nostrand Reinhold, 1994.6. Тыугу Э.Х. Концептуальное программирование. – М.: Наука, 1984. – 255 с.*