$$\downarrow$$
$$_{\mathbf{0}*|-}\forall x\mathrm{A},\ _{\mathbf{1}|-}\forall x\mathrm{A},\ _{\mathbf{1}-|}R^x_y\,\mathrm{A}\ //\ 0\{a,b\},\ 1\{a,b\}\ //\ 0\triangleright 1$$
$$\downarrow$$
$$_{\mathbf{0}*|-}\forall x\mathrm{A},\ _{\mathbf{1}|-}\forall x\mathrm{A},\ _{\mathbf{1}|-}R^x_x\,\mathrm{A},\ _{\mathbf{1}|-}R^x_y\,\mathrm{A},\ _{\mathbf{1}-|}R^x_y\,\mathrm{A}\times$$
$$//\ 0\{a,b\},\ 1\{a,b\}\ //\ 0\triangleright 1.$$

Отримали замкнене секвенційне дерево.

Висновки

На основі інтегрованого інтенсіонально-екстенсіонального підходу до побудови логічних і програмних систем досліджено композиційно-номінативні модальні та темпоральні логіки. Розглянуто властивості відношення логічного наслідку для множин формул транзиційних і темпоральних КНМЛ номінативних рівнів. На основі властивостей цього відношення побудовано секвенційні числення для транзиційних та темпоральних КНМЛ реномінативного і квантворного рівнів. Досліджено властивості пропонованих числень. Для таких числень доведено теореми коректності та повноти.

1. Нікітченко М. С. Основи математичної логіки / М. С. Нікітченко, С. С. Шкільняк. – К. : ВПЦ «Київський університет», 2006. – 246 с.
2. Шкільняк О. С. Композиційно-номінативні модальні та темпоральні логіки : семантичні властивості, секвенційні числення / О. С. Шкільняк // Наукові записки НаУКМА. Серія : Комп'ютерні науки. – 2008. – Т. 86. – С. 25–34.
3. Нікітченко М. С. Інтенсіонально-орієнтований підхід до побудови логічних систем / М. С. Нікітченко, С. С. Шкільняк // Проблеми програмування. – 2007. – № 2. – С. 15–40.
4. Непейвода Н. Н. Прикладная логика / Н. Н. Непейвода. – Новосибирск : НГУ, 2000. – 521 с.

O. Shkilnyak

SEQUENT CALCULI OF COMPOSITION NOMINATIVE MODAL AND TEMPORAL LOGICS

On the basis of the integrated intentional-extensional approach to construction of logical and program systems, composition nominative modal and temporal logics of nominative levels are investigated. Sequent calculi are constructed for such logics and soundness and completeness theorems are proved.

УДК 004.4

Гороховський С. С., Кульчицький Ю. М.

ПОРІВНЯЛЬНИЙ АНАЛІЗ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ РОЗВ'ЯЗАННЯ КОМБІНАТОРНИХ ЗАДАЧ НА СКІНЧЕННИХ ОБЛАСТЯХ З ОБМЕЖЕННЯМИ

Ефективне розв'язання комбінаторних (NP-повних, перебірних) задач було, залишається, і, найімовірніше, залишиться дисципліною, що викликає постійний інтерес теоретиків і практиків комп'ютерних обчислень на наступні десятиліття. Віднедавна набір технік для розв'язання комбінаторних задач отримав значне підсилення – програмування з обмеженнями в скінченних областях. У статті розглянуто основні поняття парадигми програмування з обмеженнями в скінченних областях, а також проведено практичне порівняння інструментальних засобів для розв'язання комбінаторних задач на скінченних областях з обмеженнями на базі популярних сьогодні обчислювальних платформ.

Вступ

Ефективне розв'язання комбінаторних задач було, залишається, і, найімовірніше, залишиться головним викликом теоретиків і практиків комп'ютерних обчислень на наступні десятиліття. Під комбінаторними задачами в цьому кон-

тексті розуміються NP-повні, тобто перебірні задачі. Як відомо, для цих задач існують лише такі алгоритми розв'язку, які мають експоненційну часову оцінку. Це означає, що при лінійному збільшенні розмірності задачі, час на її розв'язання зростає експоненційно.

Як відомо, велика кількість виробничих задач, таких як складання розкладу, виділення ресурсів, планування і оптимізація виробничого процесу, секвенція генома тощо, є NP-повними. Саме тому цінність ефективних технік та алгоритмів, що дають змогу звести час на розв'язання подібних задач до прийнятного, важко переоцінити.

Набір засобів для ефективного розв'язання комбінаторних задач значно еволюціонував за останні 20 років. Виникли як нові системи програмування, так і нові парадигми програмування для забезпечення розв'язання комбінаторних задач за прийнятний час. Одна з них, а саме програмування з обмеженнями в скінченних областях, заслуговує на особливу увагу.

Програмування з обмеженнями найкраще розглядати як програмну інфраструктуру для комбінування програмних компонент, що дають можливість отримати розв'язувачі на дереві пошуку, притаманні певній конкретній задачі. Програмування з обмеженнями є гетерогенним полем для досліджень, починаючи від математичної логіки і закінчуючи практичними застосуваннями, такими як задачі календарного планування.

Парадигма програмування з обмеженнями в скінченних областях [1] виникла наприкінці 80-х рр. для розв'язання комбінаторних задач (NP-повних задач). Вона розвинулась з логічного програмування з обмеженнями [2], яке, своєю чергою, було розширенням парадигми логічного програмування.

Дослідники [3] виділяють дві першопричини, завдяки яким розвинулось програмування з обмеженнями в скінченних областях. По-перше, це інтеграція певних технік і алгоритмів в інфраструктуру програмування з обмеженнями. Програмування з обмеженнями увібрало в себе деякі алгоритми з дисципліни дослідження операцій, а саме – специфічні алгоритми фільтрування та пошук методом гілок та границь. Деякі алгоритми були взяті з області штучного інтелекту, зокрема алгоритми забезпечення несуперечливості даних та пошук за відхиленнями. По-друге, це створення високоякісних програмних систем, що підтримують розробку програм з обмеженнями для розв'язання комбінаторних задач. Спочатку такі системи були розширеннями систем програмування для Prolog. Пізніше вони стали окремими бібліотеками для різних мов програмування, що, своєю чергою, засвідчило, що парадигма програмування з обмеженнями є незалежною від мови програмування. Нарешті, з'явилися мови програмування з обмеженнями і системи, що їх реалізують.

Можна навести деякі приклади програмних систем, що підтримують розробку програм з обмеженнями [1, 2, 3]. Система CHIP з'явилася як розширення мови програмування Prolog у 1988 році. Системи B-Prolog, ECL^{i}PS^e, Nicolog та інші також є розширеннями систем програмування для Prolog. Ilog Solver – це C++ бібліотека, яка з'явилася 1997 року і своєю появою довела, що парадигма програмування з обмеженнями є незалежною від мови програмування. Мови програмування з обмеженнями поєднують в собі символьну мову для моделювання задачі з підтримкою розв'язання задачі – це, зокрема, Claire (1996), Oz (1995, система Mozart), OPL (мова моделювання, 1999).

У статті увагу зосереджено на програмуванні з обмеженнями в скінченних областях як найновішій та найперспективнішій парадигмі програмування для розв'язання комбінаторних задач, – розглянуто основні поняття парадигми програмування з обмеженнями в скінченних областях.

Також порівнюються технологічності створення програм з обмеженнями та швидкості розв'язання задач в традиційних системах і парадигмах програмування.

Основні поняття програмування з обмеженнями в скінченних областях

Програмування з обмеженнями в скінченних областях є найновішою парадигмою програмування, що має на меті ефективне розв'язання комбінаторних задач. Розглянемо основні поняття програмування з обмеженнями в скінченних областях.

Для розуміння суті парадигми програмування з обмеженнями в скінченних областях доцільно ввести основні поняття цієї парадигми програмування.

Скінченна область – це скінченна множина невід'ємних цілих чисел.

Обмеження – це формула логіки предикатів. Ось типові приклади обмежень, які виникають в задачах у скінченних областях:

$$X = 67 \qquad X \in 0 \# 9 \qquad X = Y$$

$$X^2 - Y^2 = Z^2 \qquad X + Y + Z < U \qquad X + Y \neq 5 \cdot Z$$

$X_1, ..., X_9$ є попарно різними

Обмеження в області – це обмеження виду $X \in D$, де D є скінченною областю. Обмеження в області можуть виражати обмеження виду $X = n$, оскільки $X = n$ еквівалентно $X \in n \# n$.

Є три типи базових обмежень:

$$X = n,\ X = Y \text{ або } X \in D,$$

де D є скінченною областю.

Задача в скінченній області – це скінченний набір P обмежень вільних від кванторів, таких, що P містить обмеження в області для кожної змінної, яка наявна в обмеженнях P. Присвоєння змінної – це функція, що ставить у відповідність імені змінної деяке ціле число. Розв'язок задачі в скінченній області P – це набір присвоєнь змінних, який задовольняє усі обмеження з P. Задача в скінченній області може мати кілька розв'язків.

Розповсюдження обмежень – це механізм виведення для задач у скінченних областях, який звужує область кожної змінної. Наприклад, маючи нерівність $X = Y$ та обмеження в області $X \in 23\#100$ і $Y \in 1\#33$, розповсюдження обмежень може звузити області X і Y до $X \in 23\#32$ і $Y \in 24\#33$.

Обчислювальна архітектура розповсюдження обмежень називається простором і складається з набору розповсюджувачів зв'язаних зі сховищем даних з обмеженнями. Сховище даних з обмеженнями зберігає кон'юнкцію базових обмежень, наприклад, $X \in 0\#5 \wedge Y = 8 \wedge Z \in 13\#23$. Розповсюджувачі містять небазові обмеження, наприклад, $X < Y$ чи $X^2 + Y^2 = Z^2$. Розповсюджувач для деякого обмеження – це паралельний обчислювальний агент, який намагається звузити області змінних, які присутні в цьому обмеженні. Два розповсюджувачі, які мають спільну змінну, спілкуються виключно через сховище даних з обмеженнями.

Маючи сховище даних з обмеженнями, яке містить обмеження S і розповсюджувач, що містить обмеження P, розповсюджувач може повідомити сховищу даних з обмеженнями базове обмеження B, якщо кон'юнкція $S \wedge P$ породжує B, і B додає нову і несуперечливу інформацію до сховища. Повідомити обмеження B сховищу даних з обмеженнями S означає змінити сховище таким чином, щоб воно містило кон'юнкцію $S \wedge B$.

Операційна семантика розповсюджувача визначає, чи може він повідомити сховищу базове обмеження, чи не може.

Вимагається, щоб сховище даних з обмеженнями завжди було несуперечливим, тобто завжди має бути хоча б одне присвоєння змінної, яке задовольняє всі обмеження у сховищі.

Кажуть, що сховище даних з обмеженнями визначає змінну X, якщо сховище знає значення цієї змінної, тобто містить $X = n$, де n – ціле число.

Розповсюджувач називають суперечливим, якщо не існує жодного присвоєння змінної, яке б задовольняло сховище даних з обмеженнями і обмеження, накладене самим розповсюджувачем.

Розповсюджувач називають таким, що зазнав невдачі, як тільки його операційна семантика визначає, що він суперечливий.

Розповсюджувач називають породженим, якщо будь-яке присвоєння змінних, яке задовольняє сховище даних з обмеженнями, задовольняє і обмеження, накладене самим розповсюджувачем. Як тільки операційна семантика визначає, що розповсюджувач породжений, він перестає існувати.

Вимагається, щоб операційна семантика розповсюджувача визначала його суперечливість і породженість в останню чергу, уже після того, як сховище визначить усі змінні, наявні в розповсюджувачі.

Розповсюджувач називають стабільним, якщо він є таким, що зазнав невдачі, або його операційна семантика не може повідомити нової інформації сховищу даних з обмеженнями.

Простір називають таким, що зазнав невдачі, якщо хоча б один із його розповсюджувачів зазнав невдачі. Простір називають стабільним, якщо всі його розповсюджувачі є стабільними. Простір називають розв'язаним, якщо він не є таким, що зазнав невдачі і у нього не залишилось розповсюджувачів.

Присвоєння змінної, яке задовольняє усі обмеження сховища даних з обмеженнями і усі обмеження, накладені розповсюджувачами, називають розв'язком простору.

Для розв'язання задачі в скінченній області P завжди можна вибрати обмеження C і розв'язати дві підзадачі $P \cup \{C\}$ і $P \cup \{-C\}$. В такому випадку кажуть, що ми розподілили P за допомогою C.

Цю ідею можна застосувати і до просторів. Нехай S – стабільний простір, який не є таким, що зазнав невдачі і не є розв'язаним. Тоді обирається обмеження C і S розподіляється за допомогою C. Розподілення породжує два простори, один утворений додаванням розповсюджувача для C, а другий – додаванням розповсюджувача для $-C$.

Розподілювач – це обчислювальний агент, який реалізує стратегію розподілення. Якщо потік створює розподілювач, він блокується, поки той не виконає свою роботу. Якщо потрібен крок із розподіленням, розподілювач стає активним і генерує обмеження, за допомогою якого і буде розподілений простір.

Модель задачі – це подання деякої задачі як задачі в скінченній області. Модель визначає змінні та обмеження, що існують в задачі. Нетривіальні задачі вимагають різноманітних моделей і стратегій розподілення. Мистецтво програмування з обмеженнями полягає в побудові

моделі і стратегії розподілення, які породять найзручніше для обчислень дерево пошуку [1, 3, 6].

Докладно розглянути основні поняття програмування з обмеженнями, операційну семантику розповсюдження обмежень, стратегії і можливості розподілення обмежень, алгоритми розкриття вершин на дереві пошуку та практичні приклади побудови моделей задач в скінченних областях з обмеженнями планується в окремій статті.

Prolog проти Java проти Mozart

Для дослідження технологічності створення програм з обмеженнями та швидкості розв'язання задач в традиційних системах програмування було проведено такий експеримент. Було обрано класичну комбінаторну задачу і реалізовано її різними парадигмами програмування, після чого перевірено швидкість знаходження результатів у різних системах програмування для різної розмірності обраної задачі.

Було обрано такі парадигми програмування для дослідження: логічне програмування, процедурне програмування (імперативний стиль) та, власне, програмування з обмеженнями в скінченних областях. Для кожної з парадигм обрали яскравих представників, що давно і успішно реалізують відповідну парадигму програмування, зокрема: Prolog для логічного програмування, Java для імперативного стилю та Mozart (Oz) для програмування з обмеженнями в скінченних областях.

Далі детально розглянуто саму задачу та платформу, на якій вона розв'язувалась, а також наведено результати дослідження і проведено їх аналіз.

Задача про N ферзів. Ця задача була обрана для дослідження як класична комбінаторна.

Постановка задачі: розмістити N ферзів на шахівниці розміром $N \times N$ так, щоб жоден з ферзів не вбив іншого ферзя.

Модель задачі в термінах програмування з обмеженнями є нетривіальна. Задачу можна промоделювати так: нехай ферзі пронумеровані від 1 до N, і k-й ферзь завжди стоїть в k-ому стовпці. Для кожного ферзя i вводиться змінна R_i, що позначає, в якому рядку стоїть ферзь. Модель одразу гарантує, що два ферзі не стоять в одному стовпці. Для забезпечення того, що жодні два ферзі не в одному рядку, вводяться обмеження, що змінні $R_1, ..., R_n$ попарно різні. Для того, щоб два ферзі ніколи не стояли на одній діагоналі, потрібно ввести обмеження $R_i + (j - i) \neq R_j$ і $R_i - (j - i) \neq R_j$ для всіх i, j, таких, що $1 \leq i < j \leq N$. Це обмежен-ня еквівалентне обмеженню $R_i - i \neq R_j - j$ і $R_i + i \neq R_j + j$ для всіх i, j, таких, що $1 \leq i < j \leq N$.

Для розподілення обмежень застосовано стратегію розподілення типу «перша невдача», оскільки, застосовуючи цю стратегію розподілення, побудоване дерево пошуку менше, ніж у простій стратегії розподілення. Експериментально було також визначено, що у випадку з задачею про N ферзів, доцільніше спершу перевіряти середнє значення з області вибраної для розподілення змінної.

Програма, що написана для реалізації описаної вище моделі у термінах програмування з обмеженнями в скінченних областях на платформі Mozart, шукає всі розв'язки задачі.

Для дослідження також окремо реалізовано знаходження всіх розв'язків задачі про N ферзів в класичній парадигмі логічного програмування на Prolog і в імперативному стилі (перебірний алгоритм) в парадигмі об'єктно-орієнтованого програмування на Java.

Для зменшення похибки вимірювання часу розв'язання програми було запущено на знаходження всіх розв'язків задачі з різною кількістю ферзів (12–15) по десять разів для кожної з досліджуваних кількості ферзів, а отримані результати усереднено.

Платформа, використана для дослідження. Платформа, на якій проводили дослідження:

- Системна плата: ASUSTeK Computer Inc. P5B.
- Процесор: Intel® Core™2 CPU 6300 @ 1.86 GHz (2 процесори).
- Оперативна пам'ять: Corsair 1024 Mb DDR2 533 MHz.
- Операційна система: Microsoft Windows XP Media Center Edition, Service Pack 2.

Для дослідження використовували такі мови і системи програмування:

- Oz 3, Mozart 1.4.0 [7].
- Prolog, Visual Prolog 7.0 [9].
- Java 2 Standard Edition 1.5.0_10 [10] та NetBeans IDE 5.5 [11].

Результати дослідження. В *таблиці 1* зведено результати дослідження для різної кількості ферзів.

Таблиця 1. **Результати вимірювання середнього часу знаходження всіх розв'язків задачі про *N* ферзів різними системами програмування для різної розмірності задачі**

Кількість ферзів	Середній час, затрачений програмою на пошук всіх розв'язків задачі		
	Mozart, сек.	Prolog, сек.	Java, сек.
12	2.203	5.597	0.781
13	10.953	37.328	4.875
14	59.047	260.562	32.531
15	401.859	2296.141	– (переповнення купи)

На рис. 1 показано порівняльний графік часу, затраченого програмами на пошук усіх розв'язків задачі для різної кількості ферзів.

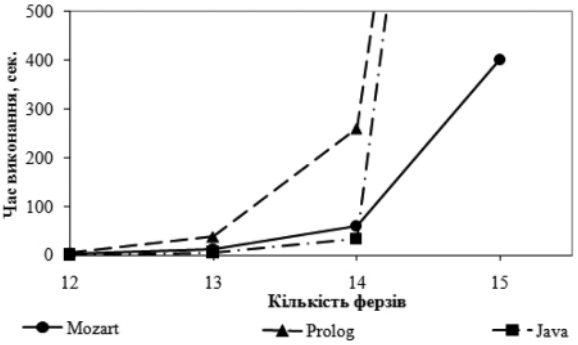

Рис. 1. Порівняльний графік часу, затраченого програмами на пошук усіх розв'язків задачі про *N* ферзів для різної кількості ферзів

Із таблиці та графіка видно, що Java лідирувала у задачах з невеликою кількістю ферзів. Коли кількість ферзів досягла 15, тобто розмірність задачі стала значною, було отримано фатальну помилку в середовищі виконання Java – склалася виняткова ситуація переповнення купи (нетипові налаштування для виділення додаткових обчислювальних ресурсів у певній системі програмування навмисне не застосовувалися для жодної із систем).

З цієї причини і для наочності рис. 1, за час роботи Java з задачею, де кількість ферзів дорівнювала 15 (де і сталася виняткова ситуація переповнення купи), взято дуже велике значення.

Щоправда, придивившись уважніше до результатів перегонів Mozart і Java, можна помітити і таку закономірність: Mozart постійно скорочував своє відставання від Java зі збільшенням кількості ферзів. Переконатися в цьому допоможе рис. 2. Так, коли кількість ферзів дорівнювала 12, Mozart був повільнішим за Java у 2,82 раза, а зі збільшеною розмірністю задачі (коли кількість ферзів дорівнювала 14) – Mozart був повільнішим лише у 1,81 раза.

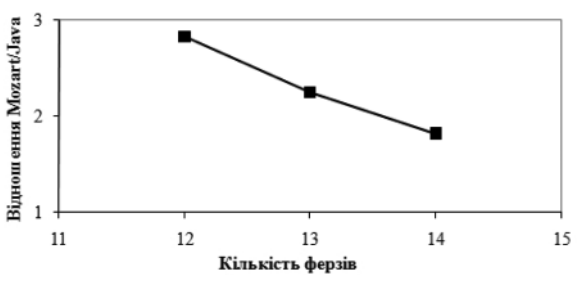

Рис. 2. Відношення середнього часу пошуку всіх розв'язків задачі про N ферзів програмою, написаною в системі Mozart, до часу пошуку всіх розв'язків програмою, написаною на Java

Отже, можна припустити, що, навіть якби у Java не склалася виняткова ситуація переповнення купи, на достатньо великій розмірності задачі Mozart би випередив Java за середнім часом, затраченим програмою на пошук всіх розв'язків відповідної задачі.

Prolog програвав і системі Mozart, і Java протягом усього змагання. На рис. 3 показано відношення середнього часу пошуку всіх розв'язків задачі, затрачуваного програмою, написаною на Prolog, до часу затрачуваного програмою, написаною в системі Mozart.

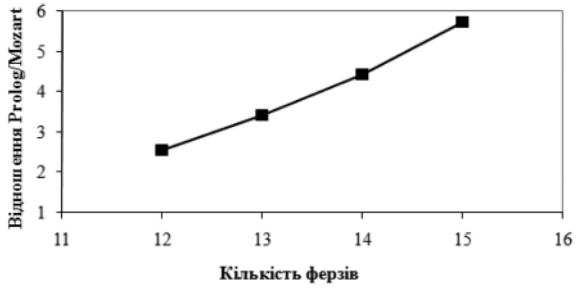

Рис. 3. Відношення середнього часу пошуку всіх розв'язків задачі про N ферзів програмою, написаною на Prolog, до часу пошуку всіх розв'язків програмою, написаною в системі Mozart

Як видно з графіку, Prolog не тільки не скорочував, а навіть збільшував своє відставання від Mozart і, відповідно, від Java зі збільшенням розмірності задачі. Так, коли кількість ферзів дорівнювала 12, Prolog був повільнішим за систему Mozart у 2,54 раза, а зі збільшеною розмірністю задачі (коли кількість ферзів становила 15) – Prolog сповільнився аж у 5,71 раза.

Висновки

Програмування з обмеженнями вважається стратегічним напрямом в дослідженнях з комп'ютерних наук [3].

Під час написання цієї роботи було розглянуто основні поняття парадигми програмування з обмеженнями в скінченних областях та наведено приклад її застосування для розв'язання класичної комбінаторної задачі – задачі про N ферзів.

Детальнішому розгляду основних понять програмування з обмеженнями та семантиці його основних операцій плануємо присвятити окрему розвідку.

У цій статті також було досліджено ефективність парадигми програмування з обмеженнями в скінченних областях та інших парадигм програмування, шляхом порівняння швидкості розв'язання задач, написаних в парадигмі програмування з обмеженнями, з іншими популярними системами програмування, що реалізують традиційні парадигми програмування.

За результатами дослідження парадигму програмування з обмеженнями в скінченних областях можна вважати мовно-незалежною, повністю придатною і ефективною парадигмою програмування для розв'язання комбінаторних задач.

Порівняльне дослідження засвідчило, що розв'язання комбінаторних задач в рамках парадигми програмування з обмеженнями в скінченних областях та її реалізації на платформі Mozart відбувається швидше і затрачає менше апаратних ресурсів обчислювальної системи, ніж розв'язання ідентичної задачі в парадигмі логічного програмування на Prolog чи в парадигмі об'єктно-орієнтованого програмування традиційним імперативним способом (перебірним алгоритмом) на Java.

Написання ж самих програм для розв'язання задач в парадигмі програмування з обмеженнями в скінченних областях в системі Mozart відрізняється від класичної програмістської практики написання коду. Тут програміст не програмує, а моделює, і лише потім програмує саму модель, користуючись стандартними чи власними мовними конструкціями. Вміння автора побудувати модель і стратегію розподілення, які створять найзручніше для обчислень дерево пошуку, для кожної задачі – в цьому, власне, полягає мистецтво програмування з обмеженнями.

Зважаючи на зрілість парадигми програмування з обмеженнями та існування її ефективних реалізацій можна виділити такі області застосування цієї парадигми, як:

- Календарне планування.
- Виділення ресурсів.
- Планування і оптимізація виробничого процесу.
- Автоматичне написання музики.
- Секвенування генома.
- Інші NP-повні комбінаторні задачі великої розмірності.

Серед можливих напрямів подальших досліджень парадигми програмування з обмеженнями можна виділити такі:

- Порівняння швидкості роботи та технологічності написання програм в парадигмі програмування з обмеженнями в системах, що реалізують цю парадигму програмування (Mozart, CHIP, B-Prolog, ECL^{i}PS^e, Nicolog, Ilog Solver та ін.)
- Порівняння парадигми програмування з обмеженнями з традиційними парадигмами програмування на інших NP-повних комбінаторних модельних задачах великої розмірності.
- Розв'язання важливої виробничої задачі, що зводиться до NP-повної комбінаторної задачі, для якої можна побудувати модель в термінах парадигми програмування з обмеженнями в скінченних областях.

1. Schulte Ch. Finite Domain Constraint Programming Systems / Christian Schulte, Mats Carlsson // In Handbook of Constraint Programming / Francesca Rossi, Peter van Beek, Toby Walsh, editors. – Elsevier, 2006. – 32 pp.
2. Letichevsky A. A Model for Interaction of Agents and Environments / Alexander Letichevsky, David Gilbert // Lecture Notes in Computer Science No 1827. Recent Trends in Algebraic Development Techniques, 2000. – P. 119–131.
3. Henz M. An Overview of Finite Domain Constraint Programming / Martin Henz, Tobias Müller // Proceedings of the Fifth Conference of the Association of Asia-Pacific Operational Research Societies. Singapore, 2000. – 9 pp.
4. Schulte C. Constraint Programming with Mozart – An Appetizer / Christian Schulte // Proceedings of Second International Mozart/Oz Conference (MOZ 2004). – Charleroi, Belgium, 2004. – 12 pp.
5. van Roy Peter. Concepts, Techniques, and Models of Computer Programming / Peter van Roy, Seif Haridi // The MIT Press, Cambridge, MA, 2004. – 904 pp.
6. Schulte C. Finite Domain Constraint Programming in Oz. A Tutorial / Christian Schulte, Gert Smolka // Tutorial, distributed with Mozart Programming System, 2008.
7. Mozart Consortium. The Mozart Programming System v. 1.4.0. – System available from official website: http://www.mozart-oz.org/; Programming Systems Lab, Saarbrücken, Swedish Institute of Computer Science, Stockholm, and Université catholique de Louvain, Louvain, 2008.
8. Haridi S. Tutorial of Oz / Seif Haridi, Nils Franzин // Tutorial, distributed with Mozart Programming System, 2008.
9. Visual Prolog v. 7.0. – System available from official website: http://www.visual-prolog.com, 2007.
10. Sun Microsystems. Java 2 Standard Edition v. 1.5.0_10. – System available from official website: http://www.java.sun.com/, 2006.
11. Sun Microsystems. NetBeans Integrated Development Environment v. 5.5. – System available from official website: http://www.netbeans.org, 2006.

S. Gorokhovsky, Yu. Kulchytsky

COMPARATIVE ANALYSIS OF TOOLS FOR SOLVING COMBINATORIAL PROBLEMS OVER FINITE DOMAINS WITH CONSTRAINTS

Effective solving of combinatorial (NP-complete, exhaustive search) problems was, is, and is most likely to remain the discipline, which provokes permanent interest of theorists and practitioners of computer calculations for the next decades. Recently, a set of techniques for solving combinatorial problems has gained considerable reinforcement – finite domain constraint programming. The paper contains basic concepts of finite domain constraint programming paradigm, as well as practical comparison of tools for solving combinatorial problems over finite domains with constraints, based on the popular modern computing platforms.