КОНЦЕПЦІЯ ДІАЛОГОВИХ ОБЧИСЛЕНЬ ТА ДЕЯКІ ПРОБЛЕМИ АВТОМАТИЗАЦІЇ ПРОГРАМУВАННЯ

Петрушенко А.М., Хохлов В.А.

Херсонський державний технічний університет, Україна, 73008, м. Херсон, Бериславське шосе, 24,
тел. 51-57-31, 55-17-31, E–mail: xvadim@newmail.ru

Розглядається концепція діалогових обчислень та наводяться деякі її раелізації в термінах алгебри алгоритміки, насамперед, уточнення основної задачі програмування. В термінах спеціального класу алгоритмів – діалогових адресних програм – дається уточнення поняття "методу програмування" та показується його зв'язок з концепцією змішаних обчислень і проблемою оптимізації програм. Пропонується одне із можливих визначень поняття "діалоговий процес". Введені поняття демонструються на прикладах. Отримані в роботі результати знаходять реалізацію в діалоговій трансформаційній машині – інструментарії діалогових алгебро-граматичних методів представлення знань.

The conception of dialogue calculation is shown. The specification of idea "method of programming" is given. The definition of concept "dialogue process" is proposed. An introduced concepts are demonstreted by examples.

Концепція діалогових обчислень та основна задача програмування

Продемонструємо суть *концепції діалогових обчислень* наступним чином [1]. Нехай ми маємо деякий *набір інструкцій H*, *єдине правило виводу – підстановку*, запис F „*умов*" задачі; необхідно з запису F вивести запис G „*рішення*" задачі. Процедура виводу розпадається на окремі кроки. На кожному кроці застосовується *інструкція – правило виводу, що складається, як видно, з двох частин – правила, що задає властивості і правила безпосередньої переробки*. З кожною властивістю ототожнюється, взагалі говорячи, декілька ділянок *конструктивного об'єкту*. Ділянка, що розглядається в даний момент, називається *активною частиною* цього перетворюваного об'єкту. Використовуючи *правило безпосередньої переробки*, що точно встановлює, як перетворювати (розчленовувати) активну частину, можливо, видаляючи при цьому деякі частини конструктивного об'єкту і вставляючи інші, необхідно отримати (вибираючи лише „головний" шлях і ігноруючи шляхи, що приводять до безперспективних слів) відповідні наслідки.

Введемо важливе поняття концепції, що пропонується, – *протоколу породження* діалогових обчислень. У цьому протоколі фіксується інформація, необхідна для однозначного відтворення процесу рішення задачі. Зрозуміло, маючи протокол породження, можна повертатися на раніше пройдені етапи процесу пошуку рішення з метою корекції цього процесу. Процес пошуку рішення продовжується ітеративно доти, поки не буде досягнутий один із заключних станів – рішення (тобто ціль, у даному випадку G, досягнута) або неуспішне завершення (ціль не досягнута).

Тут доречно нагадати, що поняття алгоритму Колмогорова [2] містить в собі *фундаментальну ідею ітеративності алгоритмічного процесу*, відповідно до якої всяке детерміноване перетворення одних об'єктів в інші представляє собою результат багаторазового застосування однієї і тієї ж, фіксованої для даного перетворення, операції – *оператора безпосередньої переробки*. Слід відзначити, що концепція дозволяє розглядати як детерміновані, так і недетерміновані моделі обчислень з пам'яттю. За рахунок умовного поділу інструкцій на дві частини: властивості задаються в *метамові*, а правила безпосередньої переробки – в *предметній*, концепція допускає явне керування перетвореннями, зокрема, дозволяє вибрати місце і послідовність інструкцій, що застосовуються. З цією взаємодією пов'язане уточнення такого фундаментального поняття комп'ютерної науки, яким є поняття алгоритму [21,26].

Твердження 1. *Запропонована концепція діалогових обчислень завдяки поняттю протоколу не тільки реалізує фундаментальну ідею ітеративності алгоритмічного процесу, але, до того ж, фіксує всі проміжні стани обчислень і дозволяє у діалоговому режимі здійснити перехід до наступного стану в процесі пошуку рішення даної задачі.*

Зауважимо, при відповідній інтерпретації понять, що входять в концепцію, яка пропонується, – конструктивний об'єкт, правило безпосередньої переробки, активна зона, F, G, H та інших, можна отримати постановки та вирішення відомих програмістських задач (перетворення формул, проектування програм, мовне процесування та інших). В залежності від класу задач, що розв'язуються, виникають відповідні протоколи діалогових обчислень. В якості приклада розглянемо спочатку уточнення так званої *основної задачі програмування* [3-9].

У найбільш загальному вигляді *основна задача програмування* може бути сформульована в такий спосіб: *по вхідній постановці задачі потрібно одержати програму в цільовій МП, що вирішує дану задачу*. Приведене формулювання потребує уточнення, оскільки майже кожне слово в ньому не є точним математичним поняттям. До того ж у цьому формулюванні нічого не говориться про сам *процес рішення* задачі. Необхідне уточнення можна отримати в такий спосіб.

Взагалі говорячи, вхідна постановка задачі формулюється в термінах предметної області (ПрО) цієї задачі. Вхідна постановка задачі повинна задаватися, як правило, у вигляді *дано-отримати*, а програма повинна обчислювати, обумовлену цими *дано-отримати*, деяку функцію *f*. *Опис* цієї функції в термінах

математичної моделі актуальної ПрО називають *головною функціональною моделлю програми* (ГФМП). ГФМП дає опис *зовнішніх* властивостей програми – які в неї існують розпорядження, що вийде в результаті їхнього виконання і так далі. Як правило, розпорядження виконує деяку дію над *об'єктом* і полягає в зміні *стану* (чи *значення*) цього об'єкта. Під *об'єктом* при цьому розуміється деяка ділянка пам'яті комп'ютера, а під його *станом* – інформація, що у даний момент часу в цій ділянці зберігається.

Зовнішній опис досить зручний для використання (поняття ГФМП близьке за змістом поняттю *специфікації*, але не тотожно йому), але в ньому нічого не говориться про його *внутрішній* устрій, оскільки *остаточний результат проектування* (текст програми) *повинен бути виражений у термінах ПрО обчислювальної системи*, за допомогою якої повинна виконуватися програма. У цілому процес переходу від вхідної постановки задачі до ГФМП не формалізований і істотно залежить від ПрО.

Основою для побудови моделі ПрО є *базова алгебра даних*, компонентами якої є *абстрактні типи даних* (різні алгебри такого типу можна отримати в результаті використання стандартних теоретико-множинних конструкцій, деякі з яких будуть розглянуті нижче). В алгебраїчних термінах *основну задачу програмування* можна переформулювати в такий спосіб: по зовнішньому опису F програми G, яку треба створити, і зовнішньому опису базової алгебри A, що вважається існуючою, необхідно знайти реалізацію $R(F,A)=G$ – текст на МП, що описує функціонування F у термінах A і що дозволяє синтезувати F при наявності A. При цьому програма $G=R(F,A)$, як мінімум, повинна бути *правильною*, *зрозумілою* і *легко виправною*.

Створення такого тексту являє собою важку задачу, для рішення якої в даний час розроблено декілька підходів [3-9]. Як правило, всі вони зводять процес проектування до послідовності етапів, що підтримують основний метод подолання складності при проектуванні великих систем – метод зведення вхідної задачі до більш простих підзадач (*аналітичний метод*). Процес зведення базується на операції "*крок декомпозиції*" $X \Rightarrow Y \& R(X,Y)$, що дозволяє по зовнішньому опису X знайти реалізацію X на базі придуманого нами *проміжного оператора* Y і одночасно зовнішній опис цього Y. Технологічний ланцюжок програмування виглядає в цьому випадку наступним чином: $F \Rightarrow B \& R(F,B) \rightarrow B \Rightarrow C \& R(B,C) \rightarrow ... \rightarrow Z \Rightarrow A \& R(Z,A)$, тобто складається в послідовному застосуванні кроку декомпозиції, починаючи з зовнішнього опису F шуканої програми G доти, поки не дійдемо до існуючої базової алгебри A. При цьому рішення основної задачі програмування – текст $R(F,A)=G$ – ми отримуємо у вигляді *композиції реалізацій*: $R(F,A)=R(F,B)*R(B,C)*...*R(Z,A)$, де B, ..., Z – проміжні оператори (їхні зовнішні описи), запропоновані в процесі рішення задачі.

Відзначимо, при проведенні кроку декомпозиції X, як правило, реалізується на базі не одного Y, а декількох. Тому у формулі для кроку декомпозиції Y варто розуміти як позначення *вектора*, тобто вірніше було б писати $X \Rightarrow (Y_1, ..., Y_n) \& R(X,(Y_1, ..., Y_n))$. Очевидно, у якості Y_i можна використовувати оператори, зовнішні описи яких були отримані раніше, на попередніх кроках декомпозиції.

Таким чином, за винятком, можливо, найпершого кроку, перед нами буде стояти не задача написання однієї окремо взятої програми, а задача реалізації одного оператора на базі якихось інших. У процесі проектування ми можемо побачити, що запропонована ієрархія проміжних операторів невдала, що якийсь оператор не реалізується чи реалізується занадто складно. У цьому випадку необхідно повертатися на декілька кроків декомпозиції назад і починати процес програмування з відповідного кроку. Приведений вище технологічний ланцюжок є *ідеалізацією* й *описує процес програмування без таких повернень*.

Алгебро-граматична точка зору на процес проектування програм полягає в тому, що в якості математичної моделі ПрО задачі використовується граматика структурного проектування (ГСП), визначена над багатоосновною (багатосортною) алгеброю структур даних. У загальному випадку, як створення ГСП, так і "витяг" з неї алгоритму для рішення задач деякого конкретного типу – у тих випадках, коли це можливо, – вимагає, узагалі говорячи, високої кваліфікації і винахідливості, глибокого проникнення в семантику актуальної ПрО. У деяких випадках, при заданій ГСП, процес "витягу" з даної моделі алгоритму стає таким, що його може в точності виконати людина, що не має ні найменшого уявлення про сутність самої задачі. Потрібно лише, щоб ця гіпотетична людина була здатна виконувати ті елементарні операції, з яких складається процес, і, крім того, щоб вона сумлінно й акуратно керувався запропонованим розпорядженням (*алгоритмом керування виводом*), закладеним у ГСП. У таких випадках людину, що, керуючись визначеним алгоритмом, одержує алгоритм рішення конкретної задачі, можна дійсно замінити *машиною*, що виконує той же процес. Характерна риса цієї машини полягає в тому, що, починаючи з того моменту, як у неї введені початкові дані (умова задачі) і програма (алгоритм, що знаходить розв'язок задачі), вона може працювати без усякого втручання людини аж до видачі остаточного результату. При цьому, звичайно, не задовольняються констатацією того, що з відповідної ГСП можна "витягти" для даного кола задач алгоритм, що розв'язує їх , а прагнуть відшукати по можливості більш "оптимальний" алгоритм.

Спроба автоматизувати ці процеси в загальних постановках задач, тобто спроба знайти їхнє повне і точне рішення, наштовхується на фундаментальні труднощі, що пов'язані з алгоритмічною нерозв'язністю. Щоб найбільш цікаві з них, насамперед у таких областях інтелектуальної діяльності людини як проектування алгоритмів і програм, їхня модифікація, зокрема, оптимізація і спеціалізація, доказ теорем і тому подібне, не були безповоротно закриті, необхідно виробити *реалістичний підхід*, заснований на відмовленні від повноти, абсолютної достовірності чи, можливо, чогось ще. Один з таких підходів дає концепція діалогових обчислень.

Дійсно, якщо F – аксіома ГСП, H – її схема (сукупність продукцій), а G – необхідний алгоритм, що формулюється в метамові неявно, у вигляді *мети*, і який необхідно одержати в предметній мові з використанням продукцій H, то концепція дає "рішення" основної задачі програмування. Оскільки процес

багаторівневого проектування алгоритмів і програм трактується у ГСП як вивід, на кожному кроці якого застосовується одне з правил граматики, то тим самим отримує уточнення операція „крок декомпозиції”.

Твердження 2. *Концепція діалогових обчислень в алгебро-граматичних термінах дає уточнення основної задачі програмування та шлях до автоматизації її рішення*.

Розглянемо алгоритмічний *клон* (теорія клонів покладена в основу *метаалгебри алгоритміки* – верхнього рівня дворівневої *алгебри алгоритміки* [4]) ALK = <{ОПЕР, L(2)}; СУПЕР>, де ОПЕР – множина неінтерпретованих операторних схем, L(2) – множина бульових функцій (БФ), СУПЕР – сигнатура, що включає лише операцію суперпозиції. Розглянемо деяку множину утворюючих Z⊂ОПЕР, в яку входять, зокрема, діалогові операції [10]. Замикання [Z] зазначеної множини по суперпозиції породжує алгоритмічний *клон діалогових схем* KDS⊂ALK: KDS = <{АДС, L(2)}; СУПЕР>, де АДС – множина діалогових неінтерпретованих схем. Нехай $Z=Z_o\cup Z_y$, де $Z_o\subset$ОПЕР, $Z_y\subset L(2)$. За допомогою приєднання операцій, які входять у Z, до сигнатури СУПЕР клону KDS може бути отримана *алгебра діалогових алгоритмів* (АДА) ADA=<{АДС, L(2)}; СУПЕР∪Z>, породжена сукупністю змінних $V=V_0\cup V_y$, де V_0 і V_y – елементарні операторні і предикатні змінні.

Розробка сукупностей інтерпретацій змінних із *V* сполучена з побудовою *багатоосновних алгебраїчних систем* (БАС) <{D_i: i∈I}; СИГН>, де $СИГН=S_o\cup S_y$. Якщо сорти D_i інтерпретувати як множини оброблюваних даних, то БАС являє собою формалізацію концепції абстрактних типів даних (АТД). З АТД асоціюється *гіпотетична машина*, доступ до станів операційної структури якої і їхнє перетворення можливі лише в результаті застосування команд ГМ – елементарних операторів і предикатів, що входять до складу сигнатури АТД. Розміщення елементів АТД на носіях абстрактних типів пам’яті (АТП) забезпечує можливість виконання тієї чи іншої програми ГМ, оформленої у вигляді регулярних схем (РС) над сигнатурою АТД.

Будемо говорити, що *схема F трансформаційно зводиться в АДА до схеми G, якщо існує вивід G із F, на кожному кроці якого застосовується одне із діалогових метаправил: згортка, розгортка (та їх сполучення) і застосування співвідношення.*

Відзначимо, що, на відміну від метаправила трансформації алгебри алгоритміки [4], діалогова трансформація розуміється в широкому значенні – не тільки перетворення з використанням співвідношень, але і проектування алгоритмів і програм, що задовольняють визначеним властивостям.

Під *діалоговою трансформаційною машиною* (ДТМ) розуміється гіпотетична машина, асоційована з концепцією АТД [11]. Список команд такої машини (видима частина АТД) базується на зображувальних засобах діалогових алгебро-граматичних формалізмів, що побудовані і досліджені у [10]. По аналогії з мовою САА\1, яка базується на апараті алгебри Глушкова, запропонована мова САА\Д проектування діалогових алгоритмів – вхідна для ДТМ, яка базується на зображувальних засобах діалогових ГСП. Побудовано апарат трансформації подібних проектів, який базується на метаправилах виводу, прийнятих в ГСП і збагачених засобами діалогу – діалогові згортка, розгортка, переінтерпретація і застосування співвідношень. Процес проектування діалогових алгоритмів і програм, оформлених засобами діалогових алгебро-граматичних формалізмів підтримуються в ДТМ спеціальною підструктурою – *машиною виводу* (МВ). Крім того, ДТМ представляє собою систему відкритого типу, яка, згідно з концепцією об’єктно-орієнтованого підходу, допускає підключення не тільки різноманітних цільових мов програмування (як це було в синтезаторі МУЛЬТИПРОЦЕСИСТ), але і різних ПрО (*баз знань*).

Методи програмування діалогових адресно-обчислюваних функцій, змішані обчислення і оптимізація програм

Як показав розвиток класичної і прикладної теорії алгоритмів, ні конкретний вибір обчислювальної моделі (ОМ), ні конкретний вибір способу програмування (при фіксованій моделі) майже (тут маються виключення, пов’язані з поняттям норми програми) не впливають на те, які теореми про обчислювальні функції і їхні програми виявляться справедливими. Іншими словами, у *широких межах можлива єдина теорія методів програмування обчислюваних функцій*. Точним математичним підтвердженням цієї тези є теорема Роджерса про ізоморфізм гьоделевих нумерацій [2]. Цей результат відкриває шляхи пошуку формального визначення поняття “метод програмування” (МеПр), яке, як відомо, в силу своєї загальності, є інтуїтивним поняттям.

Зокрема, у відповідності зі сказаним, в класичній теорії алгоритмів в якості формального визначення МеПр обчислюваних функцій пропонується використовувати *властивість гьоделевості* так званої *результатної функції* $\Re(p,x)\sim R(A(p,x,\mu t\Pi(A(p,x,t)))$ [2], де $A(p,x,t)$ – тримісна функція (її можна називати *адресною*), значення якої дорівнює (повному) стану обчислювального процесу в момент часу *t* при роботі машини з програмою $p\in P$ на вхідному даному $x\in X^{\infty}$. *Вхідна процедура* є відображення $x\to A(p,x,0)$. *Процедура закінчення* являє собою предикат П, заданий на множині всіх можливих станів обчислювального процесу, а *вихідна процедура* – відображення R цієї множини в множину Y^{∞}. Час роботи даної програми $p\in P$ на даному аргументі $x\in X^{\infty}$ визначається як найменше число *t*, при якому $A(p,x,t)$ має властивість П. Значення $R(A(p, x, t))$ при цьому *t* природно вважати *результатом роботи* програми $p\in P$ на вхідному даному $x\in X^{\infty}$.

Говорячи неформально, властивість гьоделевості й асоційовані з нею поняття (універсальна машина, *s-m-n*-теорема Кліні й інші) є абстракцією “доброго” МеПр, якщо на даному рівні абстракції “добрим” вважати такий МеПр, що за допомогою універсальної машини описує всі обчислення і дозволяє програми, складені іншим методом, ефективним образом переводити (*транслювати*) у програми, складені даним методом. Як показує практика, усі природно виникаючі МеПр мають властивість гьоделевості. Тому цю властивість можна

розглядати як уточнюючу наші інтуїтивні уявлення про МеПр подібно тому, як теза Чьорча уточнює наші інтуїтивні уявлення про обчислюваність. При неформальному розумінні МеПр тезу гьоделевості для моделі, так само як і тезу Чьорча, не можна довести в звичайному математичному розумінні, але її можна підтверджувати, розглядаючи різні представницькі ОМ і різні точно описані МеПр. Однак, якщо теза Чьорча дає точний опис класу обчислювальних функцій, у той час як тут ми маємо лише "верхню" оцінку для класу всіх МеПр.

Однак результатна функція лише частково відображає наші інтуїтивні уявлення про процес застосування даної програми до даного аргументу. Нам хотілося б, щоб весь процес обчислення, що веде від $x \in X^{\infty}$ до $y \in Y^{\infty}$, усі проміжні викладки можна було запротоколювати так, щоб цей протокол містив вичерпну інформацію про всі послідовні етапи процесу. Ці уявлення мають витоки, насамперед, у реальному процесі виконання програм на реальних ЕОМ. У зв'язку з цим опишемо деякий клас алгоритмів спеціального вигляду – *діалогові адресні програми (ДАП)*, що представляють собою послідовність пронумерованих команд, кожна з яких має один із видів(див. також [12]:

1) '*a*←*b* (*присвоєння значення*);
2) '*a*←'*b* (*пересилання чи переадресація*);
3) '*a*←'*b*+'*c* (*додавання*);
4) ЯКЩО '*a*='*b* ТО ЙТИ ДО *m* (*умовний перехід*);
5) СТОП (*зупинка програми*);
6) ПРИЗУПИНИТИ; (*тимчасова зупинка програми*);
7) ВИВЕСТИ(*x*); (*виведення на екран 'x*);
8) ВВЕСТИ(*x*); (*присвоєння 'x деякого значення b*);
9) ПРОДОВЖИТИ; (*відмінює тимчасову зупинку програми*).

Тут *a, b, c, m, n* – деякі натуральні числа, *'x – штрих-операція* [13]. У [14] команди *обнуління* (окремий випадок присвоєння значення), *додавання одиниці* і *переадресації* названі *арифметичними*.

Адресні програми можуть виконаються на *ідеальних адресних обчислювальних машинах* (ІАОМ), які мають нескінченне число пристроїв, призначених для збереження (запам'ятовування) натуральних чисел. Ці пристрої називають *комірками пам'яті чи регістрами*. Регістри забезпечуються номерами (адресами) 0, 1, 2,... у зв'язку з чим одержують наочний зміст символи команд, що були введені раніше. Переробка інформації в ІАОМ має форму встановлення і зміни деякого *адресного відображення*, відслідковувати зміни якого зручно мовою процесів (див. наступний підрозділ).

Станом ІОАМ називається нескінченна послідовність натуральних чисел $s=(s_0, s_1, ...)$, у якій майже всі (усі, крім кінцевого числа) дорівнюють 0. Якщо $s_0=0$, стан називається *заключним* (станом зупинки); якщо $s_0 \geq 1$, стан називається *робочим*, а s_0 – номером команди, що виконується. Число s_{i+1} називається *вмістом і-го* регістра. Нехай *p* – деяка ДАП, $s=(s_0, s_1, ...)$ – робочий стан. Будемо говорити, що *програма p може бути застосована до стану* s, якщо s_0 – номер однієї з команд програми *p*. Команди з номером s_0 може і не бути, наприклад, якщо s_0 занадто великий, – у цьому випадку говорять, що *програма p не може бути застосована до стану* s. Визначимо тепер деякий стан s', що називають *безпосереднім результатом застосування програми p до стану* s. Стан $s'=(s_0', s_1', ...)$ визначається в такий спосіб:

1) якщо команда з номером s_0 має вигляд '*a*←*b*, то $s_0'=s_0+1$, $s_{i+1}'=s_{i+1}$ при $i \neq a$, $s_{a+1}'=b$ (уміст усіх регістрів, крім *a-го*, не міняється, в *a-му* регістрі міститься число *b*; машина переходить до виконання наступної по порядку команди);

2) якщо команда з номером s_0 має вигляд '*a*←'*b*, то $s_0'=s_0+1$, $s_{i+1}'=s_{i+1}$ при $i \neq a$, $s_{a+1}'=s_{b+1}$;

3) якщо команда з номером s_0 має вигляд '*a*←'*b*+'*c*, то $s_0'=s_0+1$, $s_{i+1}'=s_{i+1}$ при $i \neq a$, $s_{a+1}'=s_{b+1}+s_{c+1}$;

4) якщо команда має вигляд ЯКЩО '*a*='*b* ТО ЙТИ ДО *m*, то $s_{i+1}'=s_{i+1}$ при всіх *i*; якщо $s_{a+1}=s_{b+1}$, то $s_0'=m$; якщо ж $s_{a+1}' \neq s_{b+1}$, то $s_0'=s_0+1$ (вміст регістрів не міняється; якщо вміст *a-го* регістра дорівнює вмісту *b-го* регістра, то машина переходить до виконання команди з номером *m*, в протилежному випадку – до виконання наступної команди);

5) якщо команда має вигляд СТОП, то $s_{i+1}'=s_{i+1}$ для всіх *i* і $s_0'=0$ (машина переходить у заключний стан);

6) якщо команда має вигляд ПРИЗУПИНИТИ, то $s_i'=s_i$ для всіх $i \geq 0$ (машина залишається у попередньому стані);

7) якщо команда має вигляд ВИВЕСТИ(*x*), то $s_{i+1}'=s_{i+1}$ для всіх $i \geq 0$, $s_0'=s_0+1$;

8) якщо команда має вигляд ВВЕСТИ(*x*), то '*x*←*b* (тут *b* – деяке введене значення), $s_0'=s_0+1$;

9) якщо команда має вигляд ПРОДОВЖИТИ, то $s_{i+1}'=s_{i+1}$ для всіх $i \geq 0$, $s_0'=s_0+1$.

Протоколом застосування ДАП *p* називається послідовність станів $s^0, s^1, ..., s^k$, у якій кожний наступний є результатом застосування програми *p* до попереднього, а останній стан є заключним. Стан s^0 називають *початковим станом протоколу*. На відміну від детермінованого випадку, *може існувати більше одного протоколу даної ДАП p з даним початковим станом*; такого протоколу не існує, якщо *p* не може бути застосована до s^0 чи якщо в послідовності, що виникає при багаторазовому застосуванні *p* до s^0 , не зустрічається заключний стан.

Нехай N – множина натуральних чисел, k – натуральне число. Розглянемо функцію *f*: $N^k \to N$, що визначається в такий спосіб: значення *f* на наборі $<a_1, ..., a_k>$ дорівнює *b*, якщо існує протокол застосування ДАП *p*, початковим станом якого є $(1, a_1, a_2, ..., a_k, 0, 0, ...)$, а *b* є вміст 0-го регістра в заключному стані цього протоколу. Функцію *f* будемо називати функцією, *dk-обчислюваною ДАП p* (чи просто *d-обчислюваною*, якщо

значення k ясне з контексту). Функції, dk-обчислювані ДАП, будемо називати *адресно-обчислюваними функціями k аргументів*. Очевидно, що всі адресно-обчислювані функції обчислювані; з іншого боку, усі обчислювані функції, відомі в даний час, виявляються адресно-обчислюваними (звичайно, тут необхідно враховувати розглянуте раніше зауваження про ідеалізацію). Таким чином, є підстави прийняти *гіпотезу* про збіг класів обчислюваних функцій з N^k у N і адресно-обчислюваних функцій. У цьому ж розумінні можна говорити про *методи програмування діалогових адресно-обчислювальних функцій* [12].

Властивість гьоделевості МеПр можна розглядати як теоретичне обґрунтування *ідеї автоматизації програмування* – існує автоматичний спосіб одержання з універсального алгоритму його спеціалізації. Процес переходу від універсального алгоритму до його спеціалізації було названо *змішаним обчисленням* [1,2,15-18]. Однак відповідні формулювання не дають конкретного алгоритму спеціалізації: у них враховується тільки *семантична* сторона – зв'язок програми з об'єктом, що обчислюється нею, – і майже *не* враховується синтаксична. Щоб знайти *практичні основи* для спеціалізації, уже недостатньо обмежуватися "якісною" теорією алгоритмів – необхідно визначитися з вхідною мовою змішаного обчислювача (ЗмО) – деякою *мовою специфікацій* алгоритмів, призначених для виконання на базовому обчислювачі (БО); зафіксувати (базову) МП; встановити необхідний інтерфейс між названими мовами (вони, взагалі говорячи, різні).

З метою визначення вхідної мови ЗмО знову звернемося до визначення поняття МеПр, властивість гьоделевості якого "постулює", зокрема, існування алгоритму, що дозволяє ефективно переходити від програм одного типу до програм іншого. Цей алгоритм не є конструктивним об'єктом, але оскільки всі природно виникаючі МеПр все-таки мають властивість гьоделевості, то це припускає існування визначення обчислюваності, що знаходиться в рівному положенні стосовно всіх конкретних теорій алгоритмів, що стають, таким чином, моделями цього абстрактного визначення. Перше коло досліджень показує, що початковий крок абстракції повинен складатися в пошуках визначення обчислювальності для довільних ПрО з фіксованими наборами елементарних операцій і предикатів, тобто *для абстрактних алгебраїчних систем* [15].

Відправною крапкою дослідження практичних основ спеціалізації може служити *мета спеціалізації*, що розуміється змістовно: одержання більш ефективного алгоритму, ніж початковий [1,2,15-18]. Основу способів спеціалізації у цьому випадку можуть скласти, наприклад, традиційні *методи оптимізації програм*, що базуються на результатах *потокового аналізу* [16], основною задачею якого є *схематизація базової програми* з метою одержання глобальної інформації про її семантичні властивості. Альтернативним підходом до оптимізації програм та спеціалізації служить використання формалізмів [4,17-19]. Рішення проблеми інтерфейсу між вхідною мовою ЗмО і мовою БО може бути досягнуте при використанні в якості мови ЗмО мови САА\Д-схем програм чи гіперсхем, а в якості самого ЗмО – ДТМ. На вхід ДТМ подається аксіома ГСП і множина її продукцій, на виході маємо деяку САА\Д-схему, у процесі проектування якої елементи її базису автоматично відображаються в обраній базовій мові. Інтерпретуючи частину елементів базису в цій мові і "заморожуючи" інші з них, можна побудувати *залишкову схему* і здійснити часткові обчислення на БО.

Твердження 3. *МеПр адресно-обчислювальних функцій завдяки введенню поняття протоколу обчислення адресних алгоритмів явно враховує не тільки результати обчислень, а й проміжні стани, які виникають в процесі виконання адресних алгоритмів. При цьому враховується можливість спеціалізації, що властива змішаним обчисленням, зокрема, конкретизуючому програмуванню.*

У зв'язку з отриманим результатом підкреслимо плідність поняття *іменної множини* [20], що становить важливий елемент *композиційного програмування* – складової частини *програмології* [5] – сучасної науки про алгоритми та програми. Поняття іменної множини узагальнює відомий *принцип адресності*, який був покладений в основу *адресної мови програмування* [13] – однієї з перших вітчизняних мов високого рівня.

Діалогові системи та процеси

Поняття системи, в силу своєї фундаментальності, відноситься до невизначуваних понять науки [21]. Повний опис потенційної поведінки (функціонування, роботи) системи називають *процесом* [5,6,22]. На відміну від поняття системи, яке є інтуїтивним поняттям, поняття процесу допускає точну математичну трактову. Центральне значення процесів для обчислювальних систем робить дуже бажаним точне математичне визначення поняття „(діалоговий) процес", яке ґрунтувалося б на відповідних, що піддаються прямому чи непрямому спостереженню, властивостях (обчислювальних) систем і, зокрема, охоплювало б детерміновані та недетерміновані процеси.

Під *системою* будемо розуміти сукупність взаємопов'язаних і взаємодіючих компонент, що відмежована від навколишнього середовища і взаємодіє з ним як єдине ціле [21,22]. Вважається, що система служить засобом досягнення деякої *мети* і саме цій меті підпорядкована організація всієї системи. Якщо система побудована із окремих, що віддалені одна від одної, компонент, то систему називають *розподіленою*. Систему називають *послідовною*, *паралельною* чи *діалоговою у* залежності від того, як виникають активності компонент - послідовно у часі, можуть мати місце одночасно (паралельно) чи у діалоговому режимі. Для кожного типу системи визначається відповідний тип процесу [22,23].

З кожною системою асоціюється множина (універсум) E_0 *подій*, що виникають у деякому, визначеному властивостями системи і впливом її зовнішнього оточення, порядку. Тому, перш за все, необхідно вирішити, якого роду події з поведінки системи нас цікавлять і вибрати для кожної з них відповідну назву, або *ім'я*. Множина $E \subset E_0$ імен подій, що вибирається для конкретного опису системи, називається *алфавітом системи*. Алфавіт вважається постійною, заздалегідь визначеною властивістю системи. Приймати участь у події, що

виходить за рамки алфавіту, для системи логічно неможливо. Система називається *кінцевою* (зокрема, пустою) або *нескінченною* в залежності від того, кінцевою або нескінченною є її множина подій E.

Послідовність імен подій, у яких процес приймав участь до деякого моменту часу, називається *протоколом*. Повний набір всіх можливих протоколів процесу P будемо позначати через протоколи(P). Виявляється, між кожним процесом P і парою множин (E, протоколи(P)) існує (1-1)-відповідність. Останнє твердження служить достатньою основою для їх подальшого ототожнення і наступного визначення [6,22,23]. (*Детермінований*) *процес* - це пара (B,S), де B – довільна множина символів, а S – довільна підмножина B^*, така що: 1. $<>\in S$; 2. $\forall s, t\ s\hat{}t\in S \Rightarrow s\in S$.

Після визначення (детермінованого) процесу можна дати формальне визначення деяких операцій над процесами: *префікс*, *рекурсія*, *вибір*, *паралельне виконання* і інших. Однак, визначення (детермінованого) процесу не цілком достатнє визначення поняття процесу, оскільки в ньому, зокрема, ніяк не представлена можливість недетермінованого поводження. (*Недетермінований*) *процес* [22,23] – це трійка (E, W, Q), де E – довільна множина символів (для простоти – кінцева), $W\subseteq E^*\times 2^E$, $Q\subseteq E^*$, така що:

1. $(<>, \{\})\in W$;
2. $(s\hat{}t, X)\in W \Rightarrow (s, \{\})\in W$;
3. $(s, Y)\in W \ \&\ X\subseteq Y \Rightarrow (s, X)\in W$;
4. $(s, Y)\in W \ \&\ x\in E \Rightarrow (s, X\cup\{x\})\in W \vee (s\hat{}<x>, \{\})\in W$;
5. $Q\subseteq$область(W);
6. $s\in Q \ \&\ t\in E^* \Rightarrow s\hat{}t\in Q$;
7. $s\in Q \ \&\ X\subseteq E \Rightarrow (s, X)\in W$.

Найпростіший процес, що задовольняє цьому визначенню, - це $ХАОС_E=(E, E^*\times 2^E, E^*)$. Це найбільший процес з алфавітом E, оскільки кожен елемент E^* є і його протоколом, і *розходженням*, а кожна підмножина E^* є *відмовою* після будь-якого протоколу [23].

Операції над (недетермінованими) процесами задаються описом того, як отримати три множини, що визначають їх результат, із відповідних множин їх операндів. Певна річ, необхідно показати, що результат операції задовольняє всім умовам у визначенні процесу. На щастя, це зробити не складно [23].

Дамо визначення процесу, що рекурсивно визначається за допомогою μ-оператора, а також доведемо *основну теорему про нерухому точку* [22]: $\mu X: E.F(X)=F(\mu X: E.F(X))$.

Доведення того, що $\mu X: E.F(X)$ – рішення (фактично саме недетерміноване рішення) відповідного рівняння з точністю до визначення упорядкування і μ-оператора співпадає з доведенням для детермінованого випадку із [22]. В основу доведення покладено теорію нерухомої точки по Скотту [23]. Тому, перш за все, введемо на множині процесів відношення порядку:

$(C, W1, Q1)\ [\subseteq]\ (B, W2, Q2) \Leftrightarrow (C=B \ \&\ W2\subseteq W1 \ \&\ Q2\subseteq Q1)$.

Запис $(C, W1, Q1)\ [\subseteq]\ (B, W2, Q2)$ означає, що процес (B, W2, Q2) дорівнює (C, W1, Q1) або краще його у тому розумінні, що він з меншою імовірністю розходиться і з меншою імовірністю терпить невдачу.

Очевидно, це відношення є *частковим порядком* в тому розумінні, що

1. $(E, W1, Q1)\ [\subseteq]\ (E, W1, Q1)$;
2. $(E, W1, Q1)\ [\subseteq]\ (E, W2, Q2) \ \&\ (E, W2, Q2)\ [\subseteq]\ (E, W1, Q1) \Rightarrow (E, W1, Q1)=(E, W2, Q2)$;
3. $(E, W1, Q1)\ [\subseteq]\ (E, W2, Q2) \ \&\ (E, W2, Q2)\ [\subseteq]\ (E, W3, Q3) \Rightarrow (E, W1, Q1)\ [\subseteq]\ (E, W3, Q3)$;

Ланцюгом у наведеному упорядкуванні називається нескінченна послідовність елементів $\{(E, W_0, Q_0), (E, W_1, Q_1), (E, W_2, Q_2), \ldots\}$, така, що $\forall n\geq 0\ (E, W_n, Q_n)\ [\subseteq]\ (E, W_{n+1}, Q_{n+1})$. *Границя* (*найменша верхня грань*) такого ланцюгу визначається в термінах перетинів ланцюгів, що убувають, невдач і розходжень:

$$\coprod_{n\geq 0} (E, W_n, Q_n)=(E, \bigcap_{n\geq 0} W_n, \bigcap_{n\geq 0} Q_n)$$ при умові, що $\forall n\geq 0\ W_{n+1}\subseteq W_n, Q_{n+1}\subseteq Q_n$.

Говорять, що частковий порядок *повний*, якщо в ньому є найменший елемент, а всі ланцюги мають найменшу верхню грань. Множина всіх процесів над даним алфавітом E утворює *повний частковий порядок*, так як вона задовольняє наступним законам:

1. $ХАОС_E\ [\subseteq]\ (E, W, Q)$.
2. $(E, W_i, Q_i)\ [\subseteq]\ \coprod_{i\geq 0} (E, W_i, Q_i)$;
3. $(\forall i\geq 0\ (E, W_i, Q_i)\ [\subseteq]\ (E, W, Q)) \Rightarrow (\coprod_{i\geq 0} (E, W_i, Q_i))\ [\subseteq]\ (E, W, Q)$.

В термінах границі можна сформулювати і визначення μ-оператора: $\mu X: E.F(X) = \coprod_{i\geq 0} F^i(ХАОС_E)$.

Функція F із одного повного часткового порядку в інший (або в той самий) *безперервна*, якщо вона *дистрибутивна* відносно границь всіх ланцюгів, тобто, якщо $\{(E, W_i, Q_i)|\ i\geq 0\}$ утворює ланцюг, то F *безперервна* при умові, що $F(\coprod_{i\geq 0} (E, W_i, Q_i)) = \coprod_{i\geq 0} F((E, W_i, Q_i))$.

Говорять, що функція із множини протоколів у множину протоколів *монотонна*, якщо вона зберігає відношення часткового порядку. Всі дистрибутивні функції монотонні, а, значить, монотонними будуть і всі

безперервні функції у тому розумінні, що (E, W_n, Q_n) [⊆] $(E, W_{n+1}, Q_{n+1}) \Rightarrow F(E, W_n, Q_n)$ [⊆] $F(E, W_{n+1}, Q_{n+1})$, і тому права частина попередньої рівності є границею ланцюга, що росте. По визначенню, функція F від декількох аргументів безперервна, якщо вона безперервна по кожному аргументу в окремості. Композиція безперервних функцій також безперервна. І, нарешті, будь-який вираз, що отриманий застосуванням будь-якого числа безперервних функцій до будь-якого числа і комбінації змінних, безперервний по кожній із цих змінних. Всі визначені операції над процесами (крім операції "після" / [23]) безперервні у вищезазначеному розумінні. Таким чином, якщо вираз F побудований в термінах тільки цих операцій, то він буде безперервним по X. Тепер можна довести наступне твердження.

Теорема 1 (*основна теорема про нерухому точку*) [22]: $\mu X: E.F(X)=F(\mu X: E.F(X))$.

Дійсно, $F(\mu X: E.F(X))$ = (по визначенню μ) = $F(\coprod_{i\geq 0} F^i(XAOC_E))$ = (безперервність F) = $\coprod_{i\geq 0} F(F^i(XAOC_E))$ = (по визначенню F^{i+1}) = $\coprod_{i\geq 0} F^{i+1}(XAOC_E)$ = $(XAOC_E$ [⊆] $F(XAOC_E))$ = $\coprod_{i\geq 0} F^i(XAOC_E)$ = $\mu X: E.F(X)$ (по визначенню μ).

Таким чином, повний набір всіх можливих протоколів будь-якого процесу може бути відомим нам завчасно, але до початку процесу невідомо, який саме із можливих протоколів буде реалізований - його вибір залежить від зовнішніх по відношенню до процесу факторів. *У загальному випадку перерахувати всі фактори, що впливають на функціонування систем, часто буває занадто складно (якщо взагалі можливо)*. Тому нам потрібне більш загальне визначення (діалогового) процесу, у формулювання якого повинні бути включені ще деякі додаткові вказівки, зокрема, про *можливість динамічного (діалогового) керуванням процесом обчислень*, *про результати проміжних обчислень* і так далі.

До сих пір множину подій ми не інтерпретували далі. Однак, кожній події відповідає виконання деякої *дії* із множини A дій. Важливе значення для дій в процесах має змінювання *станів*. Множину S всіх станів будемо називати *простором станів* системи. Нехай надалі $\alpha: E\to A$ – позначення дії процесу. Тоді $\forall e\in E$ $\alpha(e)=q\in A$, $q: S\to S$. Ввівши простір станів, можна розглядати в ньому послідовності станів (*траєкторій*), у яких знаходиться система в фіксовані моменти часу, а саму *систему (предметну область) визначити як клас усіх дійсно можливих траєкторій системи (предметної області)*. Очевидно, елементи будь-якої траєкторії не можуть бути довільними, оскільки поточний стан, як правило, якось пов'язаний з попередніми станами. Тому у наведеному вище визначенні системи словосполучення "*дійсно можливих*" вимагає подальшого уточнення.

У зв'язку з програмами і поставленими їм у відповідність процесами виникає питання про те, які зміни станів можуть бути безконфліктно вироблені паралельно в часі. Дати відповідь на це запитання, а також розкрити деякі інші важливі аспекти функціонування систем (наприклад, паралелізм, деталізація та інші), можна за допомогою *поняття причинності* [21-24]. Позначимо через $\prec$ відношення причина-наслідок. Нехай відношення $conflict\subseteq A\times A$ для кожної пари дій $q_1, q_2\in A$ задає, чи існує конфлікт. Процес p будемо називати *безконфліктним*, якщо $\forall e, d\in E$ $(\alpha(e),\alpha(d))\in conflict \Rightarrow e\prec d\vee d\prec e$. Кінцеві процеси визначають зміни станів, якщо тільки діям зіставлені відображення змін станів $\rho: A\to(S\to S)$ і процеси являються безконфліктними. Очевидно, процеси і їх значення є формальними описами *історій протікання процесів*.

Система позначень для опису *послідовних процесів* [23], не менш потужна, ніж регулярні вирази. Використання рекурсії робить її потужність близькою до контекстно вільних граматик, але все ж не до кінця. Це відбувається тому, що при використанні *оператору вибору* $(x: B\to G(x))$, який спочатку пропонує на вибір довільну подію x із деякої множини подій B (її називають *початковим меню* процесу), а потім веде себе як G(x), де G(x) – функція, що ставить у відповідність символу $x\in B$ процес G(x) із деякої множини процесів Y, вимагається, щоб перша подія кожної альтернативи відрізнялась від інших можливих перших подій. Якщо перша подія можлива для декількох процесів, то вибір між ними залишається недетермінованим.

Зауважимо, оператор вибору є *дистрибутивним* відносно операції *недетермінованого вибору* $P|Q: (x: B\to(P(x)|Q(x))=(x: B\to P(x))|(x: B\to Q(x))$. Якщо у цьому випадку вибір буде зроблено невірно, то процес увійде в стан *дедлоку*. Реалізація може мінімізувати ризик дедлоку, відкладаючи вибір до тих пір, доки його не зробить оточення, і вибираючи той процес із P і Q, який *не* створює дедлоку. Однак ціна такої реалізації в термінах ефективності дуже висока: P і Q повинні виконуватися паралельно до тих пір, поки оточення не вибере подію, можливу для одного процесу і неможливу для іншого (у цьому розумінні операція недетермінованого вибору виявляється не самим кращим способом об'єднання процесів, оскільки оточення повинне бути готовим до того, щоб працювати з кожним із них, в той час як робота лише з одним із них була б, скоріше всього, більш простою). *Дистрибутивним* (зліва і справа) відносно недетермінованого вибору є також *паралельний оператор*: $R||(P|Q)=(R||P)|(R||Q)$; $(P|Q)||R=(P||R)|(Q||R)$. При цьому введення паралельного оператору || дозволяє задавати мови, що не є контекстно вільними. *Рекурсивний оператор, однак, не є дистрибутивним* [23]. Дійсно, легко бачити, що, наприклад, для $P=\mu X.((a\to X)|(b\to X))$ і $Q=(\mu X.(a\to X))|(\mu X.(b\to X))$ протоколи(Q)⊆протоколи(P).

У зв'язку з наведеними вище міркуваннями введемо новий різновид оператора вибору – *оператор узагальненого (діалогового) вибору* ([22], див. також [1,10]) – $ДВ(x: B\to G(x))$, за допомогою якого оточення уже *зможе* керувати вибором між недетермінованими процесами на кожному кроці ітерації. У випадку, коли початкові події для всіх процесів, що входять в область значень Y функції G(x), різні, то $ДВ(x: B\to G(x))=(x: B\to G(x))$. Якщо деякі з процесів множини Y мають спільні початкові події, то початкове меню B розширюється

додатковими *пустими* подіями (цим подіям не відповідає жодна дія) таким чином, щоб можна було встановити (1-1)-відповідність між подіями із множини B і процесами із множини Y, а функція G(x) модифікується згідно з визначенням діалогового вибору наступним чином. Якщо під час функціонування системи зустрічається спільна для деяких процесів подія, то функціонування системи призупиняється і система переходить в стан чекання вводу деякої (пустої) події. Після введення функціонування системи продовжується з процесу, відповідного введеній події.

Різниця між операторами ДВ(P|Q), (P|Q) є досить тонкою. Їх не можна розрізнити по множинам протоколів, оскільки протокол одного із них можливий і для іншого. Однак, їх можна помістити в таке оточення, в якому (P|Q) увійде у дедлок, а ДВ(P|Q) – ні. В той же час, $\mu X.ДВ(x: B\to G(x,X)) = \mu X.(x: B\to G(x,X))$, однак ДВ(x: B→G(x,X)) дає можливість динамічного (діалогового) керуванням процесом породження будь-якого протоколу із множини протоколів рішення відповідного рекурсивного рівняння, а (x: B→G(x,X)) – ні.

Підсумовуючи наведені вище міркування ми приходимо до наступної (на даному рівні абстракції) математичної моделі для (діалогового) процесу. (*Діалоговий*) *процес* [22] - це вісімка (E, W, Q, A, S, $\prec$, α, conflict), де $W\subseteq E^*\times 2^E$, $Q\subseteq E^*$ і на них накладаються обмеження, властиві недетермінованим процесам.

Твердження 4. *Визначення діалогового процесу охоплює недетерміновані (зокрема, детерміновані) процеси і представляє інтерес як більш широка модель обчислень*.

Відзначається, що існує можливість тлумачення причинно-наслідкового відношення у визначенні системи, зокрема, як часткового порядку. Часткові порядки на кінцевих множинах однозначно характеризуються множиною елементів, безпосередньо наступних за кожним елементом. Однак для часткових порядків на нескінченних множинах це *не* справедливо. *У випадку, коли процес починає вести себе хаотично, застосовується діалог*.

Виявляється, саме оточення процесу може бути описане як процес. Це дозволяє досліджувати поведінку цілої системи, що складається з процесу і його оточення, які взаємодіють по мірі їх паралельного виконання. Зокрема, *потоки даних* (вхідних і вихідних) *і функції, що їх обробляють, можуть застосовуватися для моделювання взаємодії користувача і ДТМ* [11]: користувач породжує потік завдань до ДТМ, а ДТМ виробляє потік системних виводів як „завдань” для користувача. Взаємодію користувача і ДТМ можна визначити через взаємо рекурсивні рівності. Найменша нерухома точка тоді представляє собою потік, який задовольняє рівнянню нерухомої точки. Тим самим адекватно моделюється зворотній зв’язок і взаємодія. Якщо виникає ситуація взаємного чекання (тупик), то потік обривається.

Представлення алгоритмічних знань та керування процесом їх генерації, що прийняте в діалоговій трансформаційній машині

Представлення алгоритмічних знань в ДТМ пов’язане з описом алгоритмів за допомогою схем. Оскільки САА\Д-схеми мають однаковий синтаксис з діалоговими гіперсхеми (що являють собою, по суті, різновид ГСП з розвинутими засобами керування виводом), то це дозволяє використовувати САА\Д-схеми і побудований в [10] апарат алгебр діалогових гіперсхем (АДГС) для компактного представлення класів алгоритмів і програм та гнучкого - за рахунок введення механізму параметрів, що задаються на рівні базисних операторів і умов - керування процесом їх генерації.

В якості приклада розглядається алгоритм, що описує роботу автомату по розміну металевих монет. Припустимо, що автомат працює з п’ятьма типами монет і для кожного типу в автоматі мається декілька варіантів розміну, які він пропонує на вибір своїм клієнтам. Робота такого автомату може бути описана наступною діалоговою регулярною схемою (ДРС) в АДА σ=<***A***, ***B***; Ω>:

$$ДРМ_1=\mathbf{ІН}*_{EOP}\{\mathbf{ВВ}*_{S<MIN}(\mathbf{ДВ}(\mathbf{ОП}_1|\mathbf{ОП}_2)\vee(\mathbf{ДВ}(\mathbf{ОП}_3|\mathbf{ОП}_4)\vee\mathbf{ДВ}(\mathbf{ОП}_5|\mathbf{ОП}_6)))_{S>MAX}\},$$

де **ІН** – оператор встановлення початкового стану даних алгоритму, **ВВ** – оператор введення чергової монети для розміну, $\mathbf{ОП}_1$, ..., $\mathbf{ОП}_6$ – оператори, що реалізують алгоритми розміну монети S визначеного кількісного еквіваленту; **ЕОР** – умова, що приймає значення “істина” при зупинці роботи автомату (можливо, тимчасовій, наприклад, для вилучення або поповнення запасу монет), **ДВ** – оператор діалогового вибору [10]. Інформаційна множина ***M***, на якій задані оператори й умови, що входять у РС $ДРМ_1$, асоціюються зі значеннями параметрів, з якими працює дана РС.

Припустимо далі, що автомат буде застосовуватися в умовах, коли заздалегідь відомо, що будь-яка введена в автомат монета S задовольняє одній із попередніх умов: S<MIN, S≥MIN, S>MAX, S≤MAX, MIN≤S≤MAX. *При наведеному припущенні схема* $ДРМ_1$ *стає надлишковою*. Розглядаючи її як гіперсхему в алгебрі σ‘=<***A’***, ***B’***; Ω‘> на інформаційній множині ***M’***=***S***×***L*** (***S*** асоціюється з параметрами, що керують породженням РС), поставимо задачу генерації за $ДРМ_1$ схем у САА σ‘=<***A’***, ***B’***; Ω‘>, що відповідають вказаним умовам. Розглянемо підструктуру $\boldsymbol{M_1}\subset\boldsymbol{M}'$: ∀p∈$\boldsymbol{M_1}$ **ЕОР**(p)=η. Припустимо також, що ∀p∈$\boldsymbol{M_1}$: F(**ЕОР**, p)=**ЕОР**, а оператори, що входять у РДГС $ДРМ_1$, тотожні на підструктурі ***S***: $F(\boldsymbol{R}, p^{\Psi})=\boldsymbol{R}$ і ∀p∈***M’***: $\boldsymbol{R}^{\Psi}(p)=p^{\Psi}$.

Нехай, для визначеності, попередньо задано таку умову: будь-яка введена в автомат для розміну монета S завжди менше MIN. Даній умові відповідає підмножина $\boldsymbol{M_2}\subseteq\boldsymbol{M_1}$, на якій для будь-якої монети S умова $\alpha_1\equiv$(A<MIN) є істинною, а умова $\alpha_2\equiv$(A>MAX) – не істинною. Застосувавши РДГС $ДРМ_1$ до стану p∈$\boldsymbol{M}_2$ такого,

що $p^L=\varnothing$, одержимо на стрічці ДРС $ДРМ_1{}^1=F(ДРМ_1, p^\psi)\in \boldsymbol{A}$: $ДРМ_1{}^1 = \mathbf{ІН}*_{EOP}\{\mathbf{ВВ}^*\mathbf{ДВ}(\mathbf{ОП}_1|\mathbf{ОП}_2)\}$. Аналогічно, при попередній умові S≥MIN одержимо: $ДРМ_1{}^2 = \mathbf{ІН}*_{EOP}\{\mathbf{ВВ}^*(\mathbf{ДВ}(\mathbf{ОП}_3|\mathbf{ОП}_4)\vee \mathbf{ДВ}(\mathbf{ОП}_5|\mathbf{ОП}_6))\}$ і так далі.

Нехай $\forall p\in \boldsymbol{M}_3\subset\boldsymbol{M}_1$ $\alpha_1(p)=\alpha_2(p)=\eta$; $F(\alpha_1, p^\psi)=\alpha_1$, $F(\alpha_2, p^\psi)=\alpha_2$. Застосувавши РДГС $ДРМ_1$ до стану $p\in \boldsymbol{M}_3$, такого, що $p^L=\varnothing$, одержимо на стрічці ДРС $ДРМ_1$ у САА $\sigma=<\boldsymbol{A}, \boldsymbol{B}; \Omega>$. Отже, схеми $ДРМ_1{}^i$, де i=1, ..., 5 і сама $ДРМ_1$ належать до класу ДРС, що породжуються регулярною діалоговою гіперсхемою (РДГС) $ДРМ_1$.

Припустимо далі, що кожного типу монет в автоматі мається, відповідно, N_1, …, N_5 штук. Нехай відомо, що після видачі автоматом визначеної для даного типу кількості монет, автомат поповнює свої запаси монет даного типу із деякого запасника, після чого опрацювання вхідного потоку монет триває. Припускається, що поповнення може відбуватися не більше ніж N разів для кожного типу монет, після чого функціонування автомату (можливо тимчасово - для поповнення запасу монет із зовні) призупиняється. В нашому алгоритмі кожне поповнення моделюється шляхом деталізації операторів $\mathbf{ОП}_1$, …, $\mathbf{ОП}_6$:

$$ДРМ_1=\mathbf{ІН}*_{EOP}\{\mathbf{ВВ}*_{S<MIN}(\mathbf{ДВ}(\mathbf{ОП}_1|\mathbf{ОП}_2)\vee_{S>MAX}(\mathbf{ДВ}(\mathbf{ОП}_3|\mathbf{ОП}_4)\vee\mathbf{ДВ}(\mathbf{ОП}_5|\mathbf{ОП}_6)))\};$$

$$\mathbf{ОП}_1=\mathbf{ОПП}_1{}^*_{MOZ}(\mathbf{Е}\vee\mathbf{ОП}_7{}^*_{RN}(ДРМ_1\vee\mathbf{У})); \ldots; \mathbf{ОП}_6=\mathbf{ОПП}_6{}^*_{MOZ}(\mathbf{Е}\vee\mathbf{ОП}_7{}^*_{RN}(ДРМ_1\vee\mathbf{У})).$$

Тут $\mathbf{ОПП}_1$, ..., $\mathbf{ОПП}_6$ – оператори, що реалізують відповідний алгоритм розміну монет і віднімають кількість використаних при цьому монет визначеного типу від відповідного лічильника – S1, ..., S5 (названі лічильники, якщо їх вміст дорівнює нулю, поновлюються оператором **ІН**). Оператор $\mathbf{ОП}_7$, зокрема, підраховує кількість поновлень того чи іншого лічильника S1, ..., S5 шляхом додавання одиниці до одного з лічильників R1, ..., R5 (у початковому стані лічильники R1, ..., R5 дорівнюють нулю; їх вміст не змінюється оператором **ІН**); **Е** – тотожний оператор; **У** – оператор установки значення "істина" умови **EOP**; $\mathbf{MOZ}\equiv(S1>0)\wedge(S2>0)\wedge(S3>0)\wedge(S4>0)\wedge(S5>0)$; $\mathbf{RN}\equiv(R1<N)\wedge(R2<N)\wedge(R3<N)\wedge(R4<N)\wedge(R5<N)$.

Звернувши в одну рівність, перетворимо отриману САА\Д-схему, застосувавши до неї тричі тотожність $\mathbf{ДВ}(\mathbf{A}^*\mathbf{C}|\mathbf{B}^*\mathbf{C})=\mathbf{ДВ}(\mathbf{A}|\mathbf{B})^*\mathbf{C}$ і двічі – тотожність $(\mathbf{A}^*\mathbf{C}\vee\mathbf{B}^*\mathbf{C})_\alpha=(\mathbf{A}\vee\mathbf{B})_\alpha{}^*\mathbf{C}$.

В результаті отримаємо: $ДРМ_1=\mathbf{ІН}*_{EOP}\{\mathbf{ВВ}*_{S<MIN}(\mathbf{ДВ}(\mathbf{ОПП}_1|\mathbf{ОПП}_2)\vee_{S>MAX}(\mathbf{ДВ}(\mathbf{ОПП}_3|\mathbf{ОПП}_4)\vee \mathbf{ДВ}(\mathbf{ОПП}_5|\mathbf{ОПП}_6)))^*_{MOZ}(\mathbf{Е}\vee\mathbf{ОП}_7{}^*_{RN}(ДРМ_1\vee\mathbf{У}))\}$.

Нехай тепер MIN≤S≤MAX для всіх монет, які вводяться в автомат для розміну, а N=2. Припустимо, що **RN** є змінною етапу генерації ДРС за АДГС-схемою $ДРМ_1$. Описаній ситуації відповідає множина $\boldsymbol{M}_4\subset\boldsymbol{M}_1$ така, що $\forall p\in \boldsymbol{M}_4$ $\alpha_1(p)=\alpha_2(p)=0$, $\alpha_3(p)=\eta$, $\alpha_4(p)\neq\eta$, де $\alpha_3\equiv(\mathbf{MOZ})$, $\alpha_4\equiv(\mathbf{RN})$. Покладемо $\forall p\in \boldsymbol{M}_4$ $F(\alpha_3, p^\varphi)=\alpha_3$, $\forall \mathbf{A}\in\{\mathbf{ІН}, \mathbf{ВВ}, \mathbf{ОПП}_1, \ldots, \mathbf{ОПП}_6, \mathbf{ОП}_7, \mathbf{У}, \mathbf{Е}\}\subset \boldsymbol{A}'$ $F(\mathbf{A}, p^\varphi)=\mathbf{A}$, $\mathbf{A}^\varphi(p)=p^\varphi$. Тоді, застосувавши АДГС-схему $ДРМ_1$ до деякого $p=(p^\varphi, \varnothing)\in \boldsymbol{M}'$ отримаємо ДРС $F(ДРМ_1, p^\varphi)=$

$$=\mathbf{ІН}*_{EOP}\{\mathbf{ВВ}^*\mathbf{ДВ}(\mathbf{ОПП}_5|\mathbf{ОПП}_6)^*_{MOZ}(\mathbf{Е}\vee\mathbf{ОП}_7{}^*\mathbf{ІН}*_{EOP}\{\mathbf{ВВ}^*\mathbf{ДВ}(\mathbf{ОПП}_5|\mathbf{ОПП}_6)^*_{MOZ}(\mathbf{Е}\vee\mathbf{ОП}_7{}^*\mathbf{У})\}.$$

Перетворимо отриману ДРС, застосувавши до неї двічі співвідношення $\{\mathbf{B}^*(\mathbf{E}\vee\mathbf{У}_\beta)\}_\alpha=\{\mathbf{B}\}_{\neg\alpha}{}^*\mathbf{У}_\beta$, яке має місце при вхідних значеннях "не істинність" умови β та "істина" умови α, де **В** – оператор, який не змінює значення умови β і впливає на значення α; $\mathbf{У}_\beta$ - оператор, що встановлює значення "істина" умови β. В результаті отримаємо $F(ДРМ_1, p^\varphi)$ =

$$=\mathbf{ІН}*_{\neg MOZ}\{\mathbf{ВВ}^*\mathbf{ДВ}(\mathbf{ОПП}_5|\mathbf{ОПП}_6)\}^*\mathbf{ОП}_7{}^*\mathbf{ІН}*_{\neg MOZ}\{\mathbf{ВВ}^*\mathbf{ДВ}(\mathbf{ОПП}_5|\mathbf{ОПП}_6)\}^*\mathbf{ОП}_7{}^*\mathbf{У}.$$

Оператор **У** при такому запису алгоритму можна опустити. Ввівши умову γ, що має у початковому стані значення "не істинність", і оператор $\mathbf{У}_\gamma$, який встановлює при його другій активізації значення "істина" умови γ, $F(ДРМ_1, p^\varphi)$ можна записати у такому вигляді:

$$F(ДРМ_1, p^\varphi)=\{\mathbf{ІН}*_{\gamma}\{\mathbf{ВВ}^*\mathbf{ДВ}(\mathbf{ОПП}_5|\mathbf{ОПП}_6)\}_{\neg MOZ}{}^*\mathbf{ОП}_7{}^*\mathbf{У}_\gamma\}.$$

Алгоритм, що відповідає ДРС $ДРМ_1$ у САА $\sigma=<\boldsymbol{A}, \boldsymbol{B}; \Omega>$, можна записати у вигляді *паралельної РС* (ПРС), у якій $\alpha\equiv(S<MIN)$, $\beta\equiv(S>MAX)$, $\gamma\equiv(MIN\leq S\leq MAX)$:

$$ПРМ_1=\mathbf{ІН}*_{EOP}\{\mathbf{ВВ}^*(\underline{\alpha}\,\mathbf{ДВ}(\mathbf{ОП}_1|\mathbf{ОП}_2)\vee\underline{\beta}\,\mathbf{ДВ}(\mathbf{ОП}_3|\mathbf{ОП}_4)\vee\underline{\gamma}\,\mathbf{ДВ}(\mathbf{ОП}_5|\mathbf{ОП}_6))\}.$$

Розглянемо $ПРМ_1$ як РГС у модифікованій алгебрі гіперсхем (АГС), вважаючи визначення операторів і умов, що входять в неї, такими ж самими, як і у розглянутої раніше РДГС $ДРМ_1$. У разі задавання попередньої

умови: наперед відомо, що всі числа, що вводяться, менше від MIN, з якою пов'язана область $\boldsymbol{M}_2 \subseteq \boldsymbol{M}_1$, така, що $\forall p \in \boldsymbol{M}_2$ $\alpha(p)=1$, $\beta(p)=\gamma(p)=0$, застосувавши РГС $ПРМ_1$ до стану $p \in \boldsymbol{M}_2$, в якому $p^L=\varnothing$, одержимо на стрічці РС: $ПРМ_1^1 = \mathbf{ІН}*_{EOP} \{ \mathbf{ВВ}*ДВ(\mathbf{ОП}_1 | \mathbf{ОП}_2) \}$.

Аналогічно (розглянутим вище) виглядають РС, які отримують при задаванні на етапі генерації умов S>MAX і MIN≤S≤MAX. Запишемо РГС, що містить операції асинхронного породження синхронної диз'юнкції:

$$ПРМ_2 = \mathbf{ІН}*_{EOP} \{ \mathbf{ВВ}*(\underline{\alpha}\, ДВ(\mathbf{ОП}_1 | \mathbf{ОП}_2) \bar{\vee} \underline{\beta}\, ДВ(\mathbf{ОП}_3 | \mathbf{ОП}_4) \bar{\vee} \underline{\gamma}\, ДВ(\mathbf{ОП}_5 | \mathbf{ОП}_6)) \}.$$

Наведена РГС задає клас ПРС, що відповідають сформульованій задачі та генеруються при фіксації попередніх умов, варіанти завдання яких приведені раніше. У разі відсутності будь-яких попередніх умов РГС $ПРМ_2$ породжує ПРС $ПРМ_1$ у АГМ $\sigma=<\boldsymbol{A}, \boldsymbol{B}; \Omega>$.

Якщо припустити, що наш автомат не пропонує на вибір варіант розміну монет, то алгоритм його роботи описує РС

$$НРМ_1 = \mathbf{ІН}*_{EOP} \{ \mathbf{ВВ}*_{S<MIN} ((\mathbf{ОП}_1 | \mathbf{ОП}_2) \vee {}_{S>MAX}((\mathbf{ОП}_3 | \mathbf{ОП}_4) \vee (\mathbf{ОП}_5 | \mathbf{ОП}_6))) \},$$

в якій вибір між операторами $\mathbf{ОП}_1$ і $\mathbf{ОП}_2$, $\mathbf{ОП}_3$ і $\mathbf{ОП}_4$, $\mathbf{ОП}_5$ і $\mathbf{ОП}_6$ відбувається *недетерміновано*. По аналогії з $ДРМ_1$, $НРМ_1$ можна розглядати як РГС, що породжує клас алгоритмів, обумовлений визначеними раніше параметрами.

Твердження 5. *Використовуючи заздалегідь відомі параметри, що визначають умови застосування представленого САА\Д-схемою алгоритму, можна одержати множину конкретизованих САА\Д-схем, орієнтованих на специфіку їх використання й оптимальних для заданих умов.*

Литература

1. *Петрушенко А.Н.* О диалоговых вычислениях в алгоритмических алгебрах // Кибернетика. – 1990. – № 1. – С. 13–20.
2. *Успенский В.А., Семенов Л.Л.* Теория алгоритмов: основные открытия и приложения. – М.: Наука, 1987.– 288 с.
3. *Цейтлин Г.Е., Ющенко Е.Л.* Формализованные спецификации и трансформационный синтез программ // Кибернетика и системный анализ. – 1993. – №1. – С. 127–138.
4. *Цейтлин Г. Е.* Введение в алгоритмику. – К.: Сфера, 1998. – 310 с.
5. Редько В.Н. Основания программологии // Кибернетика и системный анализ. – 2000. - №1. – С. 33-57.
6. *Капитонова Ю.В., Летичевский А.А.* Математическая теория проектирования вычислительных систем. – М.: Наука, 1988. – 296 с.
7. *Кокорева Л.П., Перевозчикова О.Л., Ющенко Е.Л.* Диалоговые системы и представление знаний Киев: Наук. думка, 1993. – 446 с.
8. *Кушниренко А.Г., Лебедев Г.В.* Программирование для математиков: Учеб. Пособие для вузов – М.: Наука. Гл. ред. физ.–мат. лит., 1988. – 384 с.
9. *Петрушенко А.Н., Хохлов В.А., Шепетухин Е.С.* Основная задача программирования и синтез операционных устройств // Вестник Херсонского государнственного технического университета. - 2001. - № 4 (10). - С. 216-221.
10. *Петрушенко А.Н.* Алгебры диалоговых алгоритмов и гиперсхем: некоторые их свойства и приложения // Вестник Международного Соломонова университета. – 2000. – №4. – С. 110-123.
11. *Петрушенко А.Н., Хохлов В.А., Ткачев В.А., Шепетухин Е.С.* Диалоговая трансформационная машина: некоторые функциональные возможности // Проблемы программирования. - 2000. - № 1-2 (Спец. выпуск) - С. 323-334.
12. *Петрушенко А.Н.* О машинно–независимых методах программирования адресно–вычислимых функций // Вестник Херсонского государственного технического университета.. – 1998. – №1 – С.134-139.
13. *Ющенко Е.Л.* Адресное программирование. – Киев: Гостехиздат, 1963 – 288 с.
14. *Успенский В.А.* Теорема Геделя о неполноте. - М.: Наука, 1982. - 111 с.
15. *Ершов А.П.* Смешанные вычисления // В мире науки. – 1984. – №6.– С. 28–42.
16. *Касьянов А. П.* Оптимизирующие преобразования программ. – М.: Наука, 1988. – 335 с. 13.
17. *Летичевский А.А.* Смешанные вычисления и оптимизация программ // Программирование. – 1990. – №1. – С.62–77.
18. *Петрушенко А.Н.* Диалоговая трансформационная машина и смешанные вычисления // Автоматика. Автоматизация. Электротехнические комплексы и системы. - 2000. - № 1 (6). - С. 91-97.
19. *Петрушенко А.Н.* Об одном подходе к проблеме автоматизации оптимизирующих преобразований алгоритмов и программ // Кибернетика и системный анализ. – 1991. – №5. – С. 127–137.
20. *Басараб И.А., Никитченко Н.С., Редько В.Н.* Композиционные базы данных. – Киев: Либідь, 1992. – 191 с.
21. *Петрушенко А.Н.* Очерки по методологии научного познания: от математических к информационным моделям мира. – Киев: Наук. думка, 1998. – 128 с.
22. *Петрушенко А.М.* Діалогові системи та процеси: основні визначення та деякі властивості // Вестник Международного Соломонова университета. - 2004. – №1. – С. -.
23. *Хоар Ч.* Взаимодействующие последовательные процессы. – М.: Мир, 1989. – 264 с.
24. *Скотт Д.* Теория решеток, типы данных и семантика // Данные в языках программирования. - М.: Мир, 1982. – С. 25-73.
25. *Брой М.* Информатика. Основополагающее введение. В 4-х ч. / Пер. с нем. – М.: Диалог-МИФИ, 1996-1998.
26. *Петрушенко* А.Н. Понятие "алгоритм": история возникновения и формирования // Вестник Херсонского государственного технического университета.. - 1998. - № 1 (3). - С.134-138.