

УДК 681.3.06

Процик П.П., аспірант.

Композиційно-номінативний підхід до специфікації програмних систем у мові Z-Notation

У роботі продовжується дослідження семантики мови специфікації програмних систем Z-Notation на основі принципів композиційно-номінативного підходу, розпочате раніше.

Проводиться побудова транзиторної моделі програмної системи на основі формальної специфікації.

Аналізуються засоби побудови схем та композиції над ними.

Ключові слова: Z-Notation, семантика, формальна, специфікація, програма.

*E-mail: piter.protsyk@gmail.com

Статтю представив д.ф.-м.н. проф. Нікітченко М.С.

Робота продовжує дослідження семантики мови Z-Notation, розпочате в [1]. У вказаній статті був запропонований композиційно-номінативний підхід [5, 6] до трактування схем як засобу формалізації специфікацій програмних систем, заснованих на транзиторних моделях. З точки зору семантики, пропонувалось розглядати схеми як особливий засіб завдання класів іменних даних. Схеми були поділені на три класи: схеми, що задають простір станів, схеми, що задають початковий стан та схеми, що задають операції (переходи між станами). Далі за набором схем однозначно можна побудувати транзиторну модель програмної системи. А її можна досліджувати, використовуючи вже існуючий математичний апарат [8, 9]. Таким чином, був показаний один з можливих шляхів використання Z специфікацій для завдання та дослідження властивостей програмних систем.

У цій роботі представлені результати досліджень таких аспектів мови схем: засоби декларації змінних, предикатів, множин, відношень та функцій. Представлено формальне визначення специфікації програмної системи на основі принципів композиційно-номінативного підходу. Проаналізовані деякі композиції над схемами та специфічні для Z засоби завдання предикатів. Тому що, хоча, мова предикатів Z-Notation і базується на традиційній мові

P.P.Protsyk, Ph.D. student.

Composition nominative approach to specifying program systems in Z-Notation

This paper is a further development of semantics of Z-Notation specification language from compositional and nominative perspective started earlier.

Transformation from formal specification to transitional model of a program system described by this specification was built.

Means of schema composition and construction are analyzed.

Key Words: Z-Notation, formal, semantics, specification, program.

предикатів першого порядку, вона збагачена розширеними синтаксичними конструкціями. І цей аспект потребує додаткових досліджень у семантиці. Більш того, оскільки, при запропонованому підході математичний апарат досить суттєво відрізняється від традиційного, то та частина мови, яка оперує зі схемами теж вимагає роз'яснення. Ці проблеми знайшли своє відображення у представленому дослідженні.

Композиційно-номінативна семантика

Композиційна семантика – це прагматично обумовлений потребами теорії та практики підхід, одним з аспектів якого є формалізація основних програмних понять. Він пройшов перевірку часом та успішно використовувався при дослідженні різноманітних проблем теорії програмування.

Програму у композиційній семантиці на найвищому рівні абстракції можна розглядати як функцію, що відображає вхідні дані у вихідні; при залученні імен даних (номінату) – функцію над номінативними даними.

Програма – це функція, побудована з базових шляхом застосування композицій. Такий підхід до розуміння програми вказує на її структурованість і важливість вивчення процесу її побудови.

Проте, слід зауважити, що специфікації

принципово відрізняються від програм і мають інше призначення, а тому повинні мати і іншу семантичну структуру. Специфікації формалізують вимоги до програмної системи. Вони описують не конкретне обчислення, а процес обчислення в цілому, властивості процесу. Традиційно для цього використовується мова формул деякої логіки. А процес функціонування системи задається шляхом визначення станів, у яких вона може перебувати, та переходів між ними. У роботі [1] був представлений метод, згідно з яким за алгебраїчною специфікацією у Z-Notation можна побудувати транзиційну модель програмної системи на основі принципів композиційно-номінативного підходу.

Більш детальну інформацію про основні положення підходу та невизначені поняття можна знайти у джерелах [2, 3, 5, 6].

Семантика мови Z-Notation

Порядок викладання матеріалу буде співпадати з порядком викладання мови Z-Notation у роботі [4], яку традиційно вважають однією з авторитетних у цьому питанні. Зауважимо, що, враховуючи об'єм роботи, тут будуть представлені тільки деякі важливі на думку автора результати.

Розпочнемо з центральних для мови понять: множини та типу.

Множини

Поняття множини і типу, є одними із головних понять у Z-Notation. При традиційному підході у мові усе є множиною. Тому і доведення властивостей специфікацій проводиться у деякій теорії множин, яка називається базовою для мови. Часто такою базовою теорією є теорія Цермело-Френкеля (ZF). До речі, свою назву Z-Notation отримала саме від назви цієї теорії множин. Крім того, ця система аксіом цікава тим, що будь яка математична теорія може бути представлена в ній таким чином, що теореми цієї теорії стануть теоремами про множини, які виводяться з аксіом ZF. Прикладом є відома серія книг колективу математиків під загальним псевдонімом Н. Бурбаки [10], в яких побудовано замкнений виклад багатьох розділів математики на основі теорії множин Цермело-Френкеля, удосконаленої Бернайсом та Гьоделем.

Цей факт важливий для Z-Notation, ще і тому, що він гарантує можливість трактування специфікації як математичного об'єкту. А це, в свою чергу, дозволяє застосовувати для її дослідження повний спектр формальних математичних методів.

Деякі з підходів визначення семантики схем, розглядають схеми в якості стилізованої синтаксичної форми завдання множин. До такого класу підходів відноситься і представлений, з тією різницею, що схеми розглядаються як засіб декларації номінативних множин. Це дозволяє більш точно розкрити природу схеми як іменованого об'єкту. Більш того, для теорії номінативних множин існують розроблені аксіоматичні системи орієнтовані на дослідження моделей програм.

Слід згадати і про один з недоліків цього підходу. Оскільки введення схем має суто синтаксичну користь, то їхня необхідність при переході до семантики втрачається.

Типи

Мова Z-Notation відноситься до класу строго типізованих мов. Це означає, що кожен об'єкт з яким оперує мова відноситься до певного типу.

Під типом у Z розуміють іменованій набір (множину) об'єктів. У контексті типів цей набір об'єктів називають носієм типу. Носій типу або задається за допомогою декларування (перерахування можливих значень), або будується з більш простих типів за допомогою композицій конструювання. Існує ряд, так званих, атомарних, чи інакше, базових типів. До них відносяться множини натуральних N , цілих Z , дійсних чисел D та множина символів латинського алфавіту **CHAR**. Крім того, до базових типів відносяться ті, які задаються за допомогою перерахування значень – декларування. Синтаксично вони мають наступний вигляд:

$$WEEKENDS = \{Saturday, Sunday\}.$$

Тут *WEEKENDS* – це ім'я типу, а вираз справа визначає носій типу: $\{Saturday, Sunday\}$.

У Z є три способи конструювання складних типів: декартовий добуток, булеан та схеми спеціального вигляду:

Декартовим добутком типів з відповідними носіями T_1, T_2, \dots, T_n називається тип з носієм:

$$T_1 \times T_2 \times \dots \times T_n = \{(x_1, \dots, x_n) \mid x_1 \in T_1, \dots, x_n \in T_n\}.$$

Елементи декартового добутку називаються кортежами. Для Z суттєвим є те, що кортежі повинні мати не менше двох компонент.

Булеан $P(S)$ типу з носієм S – це тип з носієм, що задається множиною усіх підмножин вхідної множини S : $P(S) = \{T \mid T \subset S\}$.

Кожна схема вигляду:

$$S = [x_1 \in T_1, \dots, x_n \in T_n / P(x_1, \dots, x_n)],$$

як було показано в [1], задає множину номінативних даних. Вона визначає носій нового типу S_T з іменованими компонентами.

Кожен коректно побудований вираз, який з'являється у Z специфікації, пов'язаний з деяким конкретним типом і якщо вираз визначений, то значення виразу повинно належати носію цього типу. Кожна змінна має тип, який може бути виведений з її декларації. Існують правила для отримання типу будь-якого коректно визначеного складного виразу.

Зв'язування

Для завдання семантики схем у роботі [4] використовується поняття зв'язування. Тут наведемо лише його визначення, оскільки у композиційно-номінативній семантиці воно використовується не буде. Зв'язування задається наступним чином: якщо v_1, \dots, v_n – імена, та u_1, \dots, u_n – об'єкти типів t_1, \dots, t_n відповідно, тоді зв'язуванням називається об'єкт вигляду:

$$z = \langle v_1 \Rightarrow u_1, \dots, v_n \Rightarrow u_n \rangle,$$

з компонентами $z.v_i$ рівними u_i . Йому відповідає тип:

$$\|z\| = \|v_1 : t_1, \dots, v_n : t_n\|.$$

Можна говорити, що схема задає множину зв'язувань, які задовільняють її предикативну частину.

Як уже відзначалось, на думку автора більш адекватним поняттям для математичного представлення схеми є номінативні множини. Вони теж складаються з іменованих компонент та типізованих об'єктів, але, на відміну від визначеного поняття зв'язування, для номінативних множин існує добре розроблена математична теорія, методологічний апарат та засоби дослідження. Вивченню семантики схем з цієї позиції присвячений останній розділ.

Функції

Визначені в попередніх розділах об'єкти: множини, типи, зв'язування (номінативні множини) є фундаментальними об'єктами Z -Notation. Інші математичні об'єкти моделюються, використовуючи згадані об'єкти як базові. Відношення та функції, зокрема, задаються як підмножини декартових добутків деяких множин.

Запис $X \leftrightarrow Y$ позначає множину усіх бінарних відношень на множинах X та Y або, що те саме, множину $P(X \times Y)$ підмножин множини $X \times Y$. Отождення між бінарним відношенням та його графом в Z настільки сильне, що про них говорять, як про один об'єкт.

Функції – це спеціальний клас відношень, які для кожного значення аргументів приймають не більше одного значення. Клас усіх часткових функцій з множини X у множину Y позначимо $X \mapsto Y$. Усі тотальні, або всюди визначені, функції з множини X у множину Y позначимо $X \rightarrow Y$. Якщо, прийняти, що множина X отримана за допомогою декартового добутку n множин, то з попереднього визначення отримаємо поняття n -арної функції. Тобто функції з n аргументами.

При залученні номінативних множин вводиться спеціальний клас функцій – функцій над номінативними множинами. Якщо, ND – множина усіх номінативних даних, тоді довільну функцію вигляду $f: ND \rightarrow Y$, де Y деяка множина, називають номінативною функцією.

Предикати

Серед символів, які можуть зустрічатись у мові предикатів Z -Notation розглянемо семантику для наступних:

- \forall – квантор загальності,
- \exists – квантор існування,
- \exists_1 – квантор унікальності.

Вибір символів не випадковий. У Z -Notation квантори можна застосовувати до схем, тому побудовані таким чином предикати мають відмінну від традиційної форму та семантику. Синтаксично, вирази з кванторами мають наступний вигляд:

Квантор Schema • Predicate

Семантика виразів з кванторами така: предикат «для всіх S, Q »: $\forall S \bullet Q$, де S – схема приймає значення істини, якщо для будь яких значень змінних, введених схемою S , значення предикатів схеми S приймають істинні значення, то значення істини приймає і предикат Q . Враховуючи структуру схеми, отримаємо таке формальне визначення:

$$S = [D/P] =$$

$$[x_1 : S_1; \dots; x_n : S_n / P_1(x_1, \dots, x_n); \dots; P_m(x_1, \dots, x_n)],$$

отже:

$$\forall S \bullet Q = \forall D | P \bullet Q = \forall D \bullet P \Rightarrow Q =$$

$$= \forall x_1 \in T_1, \dots, x_n \in T_n : P_1(x_1, \dots, x_n) \wedge \dots \wedge$$

$$\wedge P_n(x_1, \dots, x_n) \Rightarrow Q(x_1, \dots, x_n).$$

Аналогічно для кванторів існування, предикат «існує S таке, що Q »: $\exists S \bullet Q$ приймає значення істини, якщо існують такі значення змінних (єдині у випадку квантора унікальності) введених схемою S , що предикати схеми S та предикат Q одночасно приймають значення істини:

$$\exists S \bullet Q = \exists D | P \bullet Q = \exists D \bullet P \wedge Q =$$

$$= \exists x_1 \in S_1 \dots \exists x_n \in S_n : P_1(x_1, \dots, x_n) \wedge \dots \wedge P_m(x_1, \dots, x_n) \wedge Q(x_1, \dots, x_n).$$

Для квантифікованих виразів виконуються наступні тотожності:

- $(\forall D | P \bullet Q) \Leftrightarrow (\forall D \bullet P \Rightarrow Q),$
- $(\exists D | P \bullet Q) \Leftrightarrow (\exists D \bullet P \wedge Q),$
- $(\exists S \bullet P) \Leftrightarrow \neg(\forall S \bullet \neg P).$

Отже бачимо, що, хоча мова предикатів у Z–Notation і збагачена додатковими засобами, при запропонованому підході на семантичному рівні їх необхідність втрачається. Аналіз цієї проблематики знаходиться у роботі [7].

Схеми

Схема – це основна синтаксична одиниця мови Z–Notation. Схема $S = [D/P]$ складається з декларативної D та предикативної P частин. Спочатку у декларативній частині вводяться змінні, а потім у предикативній на них накладаються обмеження у вигляді предикатів:

$$Schema = \left[\begin{array}{l} x_1 \in T_1, \dots, x_n \in T_n; x'_1 \in T_1, \dots, x'_n \in T_n \\ \hline Q(x_1, \dots, x_n); \\ P(x_1, \dots, x_n, x'_1, \dots, x'_n); \\ R(x'_1, \dots, x'_n) \end{array} \right.$$

Змінні, які декларуються у схемі, можуть носити різне призначення: змінні станів, вхідні змінні, вихідні змінні. Якщо схема задає перехід з одного стану в інший, то для позначення змінних наступного стану, їх записують з символом ($'$).

У Z–Notation визначені композиції, які дозволяють будувати з простіших схем більш складні. Найбільш поширені з них – це композиції декорації та перейменування. Композиції задаються для схем з сумісними за типами деклараціями змінних. Дві декларації вважають сумісними за типами, якщо кожна змінна, яка присутня в обох, має однаковий тип в обох деклараціях.

Наприклад, наступні дві декларації є сумісними за типом, оскільки їх єдина спільна змінна b має однаковий тип:

$$D_1 = [a: P(X); b: X \times Y] \text{ та } D_2 = [b: X \times Y; c: Z].$$

Ці дві декларації можуть бути об'єднані, та утворити розширену декларацію:

$$D_3 = D_1 \cup D_2 = [a: P(X); b: X \times Y; c: Z].$$

При цьому, попередні декларації D_1 та D_2 називають під-деклараціями декларації D_3 . Кожна з декларацій D_1, D_2 може бути отримана з D_3 за допомогою операції обмеження (проекції) по деякій змінній.

Над деклараціями змінних у Z–Notation визначена операція декорації – ($'$) яка природним чином поширюється на схеми: якщо S – схема, тоді S' отримана за схемою S заміною імен на ті ж імена з суфіксом $'$. Тобто, декларація S' містить компоненту x' для кожної компоненти x з декларації S , при чому їх типи співпадають.

У Z існує три стандартних оформлення змінних, що використовуються в описі даних:

- ($'$) – для позначення наступного стану,
- ($?$) – для позначення вхідних змінних,
- ($!$) – для позначення вихідних змінних.

Ще одна з типових операцій на схемах – це операція перейменування. Якщо S схема, то схема $S [y_1 / x_1, \dots, y_n / x_n]$ отримана заміною кожної компоненти x_i відповідною y_i . Для того, щоб запис мав зміст, імена x_i не повинні повторюватись, а також потрібно враховувати узгодженість типів після перейменування (довільні дві компоненти з однаковими іменами повинні мати один тип).

Дві схеми S та T з сумісними деклараціями можуть бути об'єднані оператором об'єднання схем, та утворити нову схему $S \wedge T$. Декларацією змінних цієї нової схеми є об'єднана декларація змінних схем S та T , а її предикативна частина – об'єднання предикативних частин схеми S та T .

Розглянемо спрощений для наочності варіант об'єднання схем зв'язкою кон'юнкції, коли в їх деклараціях імена змінних усі різні:

$$S = [D1/P1] = [x_1: S_1; \dots; x_n: S_n / P_1(x_1, \dots, x_n); \dots; P_m(x_1, \dots, x_n)],$$

$$T = [D2 / P2] = [y_1: T_1; \dots; y_m: T_m / Q_1(y_1, \dots, y_n); \dots; Q_k(y_1, \dots, y_n)].$$

Тоді,

$$S \wedge T = [D1 \cup D2 / P1 \wedge P2] = [x_1: S_1; \dots; x_n: S_n, y_1: T_1; \dots; y_m: T_m / P_1(x_1, \dots, x_n); \dots; P_m(x_1, \dots, x_n), Q_1(y_1, \dots, y_n); \dots; Q_k(y_1, \dots, y_n)].$$

Іншими логічними зв'язками для сумісних за деклараціями схем є \vee , \Rightarrow та \Leftrightarrow . Вони визначаються аналогічним чином, при цьому, їх декларації об'єднуються, а до властивостей схем застосовується відповідна логічна зв'язка. Можна також визначити операцію заперечення схеми, при цьому сигнатура зберігається, а властивість є

запереченням відповідної властивості вхідної схеми.

Розглянемо дві важливих композиції над схемами, які використовуються при специфікації послідовних систем: композиція послідовного виконання схем (;) та композиція суперпозиції (>>).

Композиція послідовного виконання схем (;) визначається над схемами операцій, що містять спільні імена у декларації:

$$S = [x_1, \dots, x_n, x'_1, \dots, x'_n \mid P(x_1, \dots, x_n, x'_1, \dots, x'_n)],$$

$$T = [y_1, \dots, y_k, y'_1, \dots, y'_k \mid Q(y_1, \dots, y_k, y'_1, \dots, y'_k)]$$

За виконання синтаксичної рівності символів $x_{i_j} \equiv y_{p_j}$, при $j = \overline{1..m}$ та $1 \leq m \leq \min(n, k)$ і входженні змінної x_{i_j} у схему S з декорацією (').

Тоді

$$S;T = [x_1, \dots, x_n, y_{p_{m+1}}, \dots, y_{p_k},$$

$$x_{i_{m+1}}', \dots, x_{i_n}', y_1', \dots, y_k']$$

$$\exists x_{i_1}' \dots \exists x_{i_m}': (x_{i_j}' = y_{p_j}, j = \overline{1..m}) \wedge$$

$$\wedge P(x_1, \dots, x_n, x'_1, \dots, x'_n) \wedge$$

$$\wedge Q(y_1, \dots, y_k, y'_1, \dots, y'_k)].$$

Що по суті являє собою зв'язування спільних змінних, які зазнали зміни у схемі S та відповідних їм за іменами початкових змінних схеми T .

Композиція суперпозиції (>>) діє аналогічним чином, тільки вона застосовується для вихідних та вхідних змінних схем відповідно, дозволяючи передавати результати дії однієї схеми операції на вхід іншої операції. Розглянемо для прикладу варіант з однією змінною:

$$S = [x_1, \dots, x_n, x'_1, \dots, x'_n, a? \mid P(x_1, \dots, x_n, x'_1, \dots, x'_n, a?)],$$

$$T = [x_1, \dots, x_n, x'_1, \dots, x'_n, a! \mid Q(x_1, \dots, x_n, x'_1, \dots, x'_n, a!)]$$

Тоді

$$S \gg T = [x_1, \dots, x_n, x'_1, \dots, x'_n \mid$$

$$\exists a: P(x_1, \dots, x_n, x'_1, \dots, x'_n, a) \wedge$$

$$\wedge Q(x_1, \dots, x_n, x'_1, \dots, x'_n, a)].$$

Вказані композиції дозволяють будувати складні схеми-операції з більш простих. Перехід з одного стану в інший під дією побудованої таким чином схеми відбувається атомарно.

Для повноти викладу зазначимо, що в залежності від практичних потреб у мову можна також ввести і інші композиції та операції над схемами.

Мова предикатів, яка застосовується у Z-Notation подібна до класичної мови предикатів

першого порядку. Однак, є певні відмінності пов'язані із використанням схем та ряд синтаксичних скорочень, яких не має у класичному варіанті. Приклади таких скорочень були представлені у попередньому розділі.

Специфікація програмних систем у Z-Notation

Специфікацією програмної системи у Z-Notation називається набір схем, який складається з однієї або декількох схем станів, схем, що задають початкові стани та схем операцій.

Розглядається синтаксично спрощений варіант схем у такій формі:

$$\text{StateSchemaText} = [x_1, \dots, x_n \mid P(x_1, \dots, x_n)];$$

$$\text{InitSchemaText} = [x_1', \dots, x_n' \mid \text{Init}(x_1', \dots, x_n')];$$

$$\text{OpSchemaText} = [x_1, \dots, x_n, x_1', \dots, x_n' \mid$$

$$\text{Pre}(x_1, \dots, x_n);$$

$$\text{Op}(x_1, \dots, x_n, x_1', \dots, x_n');$$

$$\text{Post}(x_1', \dots, x_n')];$$

Мова розглядається у без типовому варіанті з фіксованою множиною можливих імен: x_1, \dots, x_m . Вважається, що змінні приймають значення з деякої фіксованої множини значень U . Під множиною усіх номінативних даних ND розуміємо множину номінативних даних індуковану цими іменами та множиною U так, як це вводиться у роботі [2].

Визначення. Синтаксичною специфікацією вимог до програмної системи у Z назвемо впорядкований текст, що складається з довільного скінченного числа схем станів: ST_1, \dots, ST_k ; однієї або декількох схем ініціалізації: I_1, \dots, I_m ; довільного скінченного числа схем операцій: Op_1, \dots, Op_n та довільної кількості допоміжних схем-визначень: C_1, \dots, C_p .

Схеми визначення вводяться як синтаксичні скорочення, та при переході до семантики вони об'єднуються зі схемою, в якій вони використовуються.

Введемо операцію синтаксичного об'єднання двох схем \cup :

Визначення 4.1. Нехай $A = [a_1, \dots, a_n \mid P(a_1, \dots, a_n)]$ та $B = [b_1, \dots, b_k \mid Q(b_1, \dots, b_k)]$ дві схеми, тоді їх синтаксичним об'єднанням $A \cup B$ назвемо схему побудовану таким чином:

$$A \cup B = [c_1, \dots, c_m \mid P(c_{i_1}, \dots, c_{i_n}), Q(c_{j_1}, \dots, c_{j_k})],$$

де $\{c_1, \dots, c_m\} = \text{Names}(A \cup B) = \text{Names}(A) \cup \text{Names}(B)$, та $c_{i_p} \equiv a_p, p = \overline{1, n}; c_{j_q} \equiv b_q, q = \overline{1, k}$.

Тут символ (\equiv) розуміється як рівність синтаксичних символів.

Задамо семантику схем. Кожна схема Sch буде визначати часткову функцію над номінативними даними задану своїм графіком у випадку операції або клас номінативних множин $S(Sch)$ такий, що кожен елемент (пара у випадку операції) цього класу буде задовольняти умовам специфікованим у відповідній схемі. Будемо говорити, що клас $S(Sch)$ індукується схемою Sch .

Визначення. Нехай Sch – схема, тоді вона індукує клас номінативних даних, заданий наступним чином:

- 1) Якщо $Sch = [x_1, \dots, x_n | P(x_1, \dots, x_n)]$, тоді:
 $S(Sch) = \{st | P(st)\}$, де $st \in ND$ – множина усіх номінативних даних, та

$$P(st) = \begin{cases} P(v_1, \dots, v_n), & \text{якщо } Names(Sch) \subseteq names(st) \\ & \text{та } st = [x_1 \rightarrow v_1, \dots, x_n \rightarrow v_n, \dots]; \\ false, & \text{інакше.} \end{cases}$$

- 2) Якщо $Sch = [x_1', \dots, x_k' | I(x_1', \dots, x_k')]$, тоді:
 $S(Sch) = \{st | I(st)\}$, де $st \in ND$ та

$$I(st) = \begin{cases} I(v_1, \dots, v_k), & \text{якщо } st = [x_1 \rightarrow v_1, \dots, x_k \rightarrow v_k, \dots]; \\ false, & \text{інакше.} \end{cases}$$

- 3) Якщо $Sch = [x_1, \dots, x_n, x_1', \dots, x_n' | P(x_1, \dots, x_n),$
 $Op(x_1, \dots, x_n, x_1', \dots, x_n'), Q(x_1', \dots, x_n')]$,
тоді:

$$S(Sch) = \{(st, st') | P(st) \wedge Op(st, st') \wedge Q(st')\},$$

де $st, st' \in ND$ та $P(st), Q(st')$ – визначаються аналогічно $P(st)$ з випадку 1, а

$$Op(st, st') = \begin{cases} Op(v_1, \dots, v_n, u_1, \dots, u_n), & \text{якщо } st = [x_1 \rightarrow v_1, \dots, x_n \rightarrow v_n, \dots]; \\ & st' = st \forall [x_1 \rightarrow u_1, \dots, x_n \rightarrow u_n] \\ false, & \text{інакше.} \end{cases}$$

Визначення Моделлю синтаксичної Z специфікації, (або моделлю яку індукує специфікація) будемо називати трійку, що визначається наступним чином:

$$Model = \left(\begin{array}{l} S(ST_1 \cup ST_2 \cup \dots \cup ST_k); \\ S(I_1 \cup I_2 \cup \dots \cup I_m); \\ S(Op_1) \cup S(Op_2) \cup \dots \cup S(Op_p) \end{array} \right) = (ST, I, OP).$$

Зрозуміло, що не кожна специфікація визначає програмну систему здатну щось обчислювати. Тому введемо важливе для подальшого розгляду поняття коректної моделі та специфікації.

Визначення. Специфікація називається коректною, якщо модель яку вона індукує задовольняє наступним умовам

- $ST \neq \emptyset$, тобто простір станів не вироджений;
- $I \subseteq ST$ та $I \neq \emptyset$, – існує хоча б один початковий стан;
- $\forall st \in ST : OP(st) \downarrow \Rightarrow OP(st) \subseteq ST$, – дія операцій системи не виводить за межі простору станів.

Для того, щоб існував початковий стан для специфікації $Spec = (ST, I, OP, C)$ необхідне виконання наступної теореми в Z :

$\exists State' \bullet Initial$, або інакше:

$$\exists x_1' \dots \exists x_n' : State(x_1', \dots, x_n') \wedge Init(x_1', \dots, x_n').$$

де

$$State = ST_1 \cup ST_2 \cup \dots \cup ST_k = [x_1, \dots, x_n | State(x_1, \dots, x_n)],$$

$$Init = I_1 \cup I_2 \cup \dots \cup I_m = [x_1', \dots, x_n' | Init(x_1', \dots, x_n')].$$

На семантичному рівні це твердження буде мати такий загальний вигляд: $\exists st \in ST : st \in I$.

Поведінкою системи назвемо послідовність станів $st_0, st_1, \dots, st_l, \dots$, яка починається з деякого початкового стану $st_0 \in I$ та

$$\forall k > 0 : \exists st_{k-1}, st_k : (st_{k-1}, st_k) \in OP.$$

Сформулюємо як гіпотезу таке важливе твердження: кожна коректна Z специфікація задає транзиторну систему, і навпаки, кожна транзиторна система має своє подання у вигляді синтаксичної Z специфікації.

Висновки

У роботі був представлений виклад деяких важливих аспектів мови специфікації Z -Notation на основі принципів композиційно-номінативного підходу. Були приведені загальні визначення основних понять та конструкцій мови. Продемонстровані базові композиції над схемами та їх визначення у запропонованому підході.

Сформульоване формальне визначення програмної системи заданої Z специфікацією.

Основна відмінність, від існуючих підходів про визначення семантики схем, полягає у виборі математичної теорії. В якості математичних об'єктів представлення для схем виступають класи номінативних (іменних) множин, номінативні функції та предикати, а композиції над схемами – це композиції над номінативними даними, або номінативними функціями у випадку операцій.

Виконана робота показує адекватність застосування композиційно–номінативного підходу для опису семантики цього формального методу специфікації програмних систем.

Схеми розглядаються як об'єкти синтаксичного рівня. Інші підходи та пов'язані з ними проблеми аналізуються у [7]. Кожен з підходів має свої переваги та недоліки. Однак, варто зазначити, що на даний момент не існує

єдиного домінуючого методу формалізації семантики Z.

Подальші дослідження будуть полягати у розробці методів для аналізу властивостей специфікацій та їх автоматизованого доведення. Розгляду розширених композицій над схемами для специфікації паралельних та розподілених систем.

Список використаних джерел

1. *Процик П.П.* Композиційно–номінативний підхід до побудови семантики Z–Notation // Вісник кийського університету. Сер.: Фізико–математичні науки. – 2008. – №1. – С. 116–120.
2. *Никитченко Н.С.* Композиционно–номінативный подход к уточнению понятия программы. // Проблемы программирования. – 1999.–№1. С. 16–31.
3. *Нікітченко М.С., Шкільняк С.С.,* Математична логіка та теорія алгоритмів, навчальний посібник. – Київ: ВПЦ «Київський Університет», 2007. – 164 с.
4. *J.M. Spivey.* The Z Notation: A Reference Manual.–Oxford Oriell College, 1998.– 158pp.
5. *Басараб И.А., Никитченко Н.С., Редько В.Н.* Композиционные базы данных. – Київ: ЛИБІДЬ, 1992. – 192 с.
6. *Nikitchenko N.S.* Composition Nominative Approach to Program Semantics. – Technical University of Denmark, 1998.– 103 pp.
7. *Martin Andrew* Relating Z and First–Order Logic // Journal of Formal Methods. – 1999. – Volume II. P. – 1266 – 1280 p.
8. *Clarke, Edmund M., Orna Grumberg, and Doron A. Peled* Model Checking.– Cambridge: MIT Press, 1999. – 330 p.
9. *Berard, Beatrice, Michel Bidoit, Alain Finkel, Francois Laroussinie, Antoine Petit, Laure Petrucci, Philippe Schnoebelen, and Pierre Mckenzie.* Systems and Software Verification: Model–Checking Techniques and Tools.– Berlin–Heidelberg: Springer Verlag, 2001. – 196 p.
10. *Н. Бурбаки,* Элементы математики.– Москва: Начала Математики, 1965.– 456с.

Надійшла до редколегії 18.02.09