

## МЕТОДИКА АВТОМАТИЗОВАНОЇ ТРАНСФОРМАЦІЇ ФОРМАЛЬНИХ СПЕЦИФІКАЦІЙ У ПРОГРАМНИЙ КОД, ЩО ВИКОНУЄТЬСЯ

*Київський Національний Університет ім. Т.Г. Шевченка,  
факультет кібернетики, кафедра Теорії та Технології Програмування*

Робота присвячена дослідженню процесу переходу від формальної специфікації до програми, що її реалізує. Розроблено метод, на основі якого можна перетворювати формальні специфікації, записані мовою Z-Notation [1], у код на мові програмування. В якості такої мови програмування буде використовуватись мова Script.Net [2], створена на основі принципів композиційно-номінативного підходу [3]. Цей метод допускає можливість побудови автоматизованої процедури трансформації специфікації у код, що виконується.

Специфікації описують вимоги до програми у вигляді властивостей, які у подальшому підлягають перевірці (верифікації). Формальні специфікації – це специфікації, засновані на формальних математичних теоріях. При цьому, програма трактується як математичний об'єкт. Розробку програм за формальною специфікацією можна уявляти собі як процес послідовного уточнення. Із специфікації можна виділити так звану частину, що виконується (арифметичні вирази, тощо), та декларативну. Виконувана частина буде мати пряме відображення у мову програмування. А декларативну необхідно якимось чином промодельювати засобами мови програмування. В обраній мові програмування для цього використовується механізм контрактів у формі перед-, пост- та інваріантних умов.

В якості мови специфікації використовувалась відома мова Z-Notation. Вона виникла на початку 70-х років та зарекомендувала себе при розробці послідовних систем. Існує стандарт мови та відомі успішні її використання на практиці. Структурно Z-Notation складається з мови логіки першого порядку для запису предикатів та мови схем. Мова схем використовується для структурування специфікацій, побудови визначень змінних, типів даних, предикатів та операцій.

Найбільш широко мова Z використовується для специфікації систем, заснованих на транзитивних моделях. У рамках згаданої моделі системи описуються в термінах станів та переходів з одного стану в інший. На практиці використовуються різні рівні деталізації поняття стану та переходу. Функціонування системи починається з деякого початкового стану та продовжується (імовірно, нескінченно довго), поки існує можливість переходу в інший стан. Іноді виділяють множину станів, яку називають заключною, і при переході системи в один з таких станів її виконання припиняється.

Синтаксично схема складається з двох частин: декларативної – у якій описуються змінні та предикативної – у якій на змінні накладаються обмеження у вигляді предикатів: *Схема* = [*Декларативна* | *Предикати*]. Якщо у декларативній частині змінна записана з символом (*'*), то це означає, що при застосуванні схеми значення змінної зміниться.

В залежності від типу змінних та предикатів будемо розглядати такі три типи схем:

- Схеми, що задають простір допустимих станів – *StateSchemaText* = [*x*<sub>1</sub> , ..., *x*<sub>*n*</sub> | *P*(*x*<sub>1</sub>, ..., *x*<sub>*n*</sub>) ];
- Схеми, що задають початкові стани – *InitSchemaText* = [*x*<sub>1</sub>' , ..., *x*<sub>*n*</sub>' | *Init*(*x*<sub>1</sub>' , ..., *x*<sub>*n*</sub>') ];
- Схеми, що задають операції – *OpSchemaText* = [*x*<sub>1</sub>, ..., *x*<sub>*n*</sub>, *x*<sub>1</sub>' , ..., *x*<sub>*n*</sub>' | *Pre*(*x*<sub>1</sub>, ..., *x*<sub>*n*</sub>); *Op*(*x*<sub>1</sub> , ..., *x*<sub>*n*</sub>, *x*<sub>1</sub>' , ..., *x*<sub>*n*</sub>') ; *Post*(*x*<sub>1</sub>' , ..., *x*<sub>*n*</sub>') ];

Специфікацією програмної системи у Z-Notation будемо називати четвірку: *Spec*=(*ST*,*I*,*OP*,*C*), де *ST*={ *ST*<sub>1</sub>, ..., *ST*<sub>*k*</sub> } – множина схем станів, *I*={ *I*<sub>1</sub>, ..., *I*<sub>*m*</sub> } – множина схем ініціалізації, *OP*={ *Op*<sub>1</sub>, ..., *Op*<sub>*n*</sub> } – множина схем операцій, *C*={ *C*<sub>1</sub>, ..., *C*<sub>*p*</sub> } – множина допоміжних схем.

Кожна специфікація у такій формі може бути перетворена у програму на мові Script.NET. Змінні описані у схемах першого типу перетворюються у змінні стану програми. Вхідні данні для

програми повинні задовольняти умовам накладеним у схемах другого типу. Кожна схема третього типу визначає функцію, що перетворює значення змінних. Для опису властивостей змінних у мові існує засіб специфікації предикатів у формі перед-, пост- та інваріантних умов, що носить назву контракту. Умови контракту обчислюються та перевіряються під час виконання програми. Таким чином досягається відповідність властивостей описаних у специфікації властивостям програми яка нею індукується.

Для того, щоб побудувати такий перехід необхідно, щоб семантика мови специфікації та семантика мови програмуванні були побудовані на єдиному математичному апараті. В якості такого апарату використовується композиційно-номінативного підхід. Це прагматично обумовлений потребами теорії та практики підхід, одним з аспектів якого є формалізація основних програмних понять, дослідження програм з позиції їх функціональності, іменованій природі та засобам їх побудови – композиціям. Семантика мови Z-Notation на основі композиційно-номінативного підходу представлена у роботі [4]. Пропонується розглядати схеми станів, як засіб завдання класів номінативних множин, які є даними у мові Script.NET. Схеми операції – як засіб декларативного опису функцій над номінативними даними. Композиції над схемами – як композиції над номінативними даними або функціями.

Розглянемо типовий приклад схеми, що задає операцію та функцію яка може бути отримана за нею:

Схема:  $[x:Z, x':Z \mid x > 0; x' \bmod 2 == 0]$ .

Функція:  $function f(x) [pre(x > 0); post(x \% 2 == 0); invariant(x \text{ is int}); ] \{ \text{обчислення} \}$ .

Отримана за специфікацією програма не містить тексту обчислення, вона є шаблоном для подальшого уточнення розробником. Але цей шаблон дозволяє будувати програми, які задовольняють вимогам накладеним у специфікації.

Висновки:

1. Розглянуто семантику мови Z-Notation на основі композиційно-номінативного підходу та її зв'язок з мовами програмування.
2. Побудовано метод перетворення формальних специфікації у програмний код, що виконується.
3. Реалізовано мову програмування засновану на композиційно-номінативних принципах.

Використана література:

1. *Jim Woodcock, Jim Davies: Using Z. Specification, Refinement, and Proof* –New York: Prentice Hall, 1996. – 523 p.
2. Процик П.П., Композиційно-номінативна мова програмування Script.NET // “Проблеми програмування”. № 2-3 (спеціальний випуск). Київ – 2008, 323-331 с.
3. Никитченко Н.С. Композиционно-номинативный подход к уточнению понятия программы. // Проблемы программирования.– 1999.–№1. С. 16– 31.
4. Процик П.П. Композиційно-номінативний підхід до побудови семантики мови специфікації Z-Notation // Вісник КНУ, фізико-математичні науки, вип. №1, 2008.