

Процик Петро Павлович,
Кафедра Теорії та технології програмування
Київського національного університету ім. Тараса Шевченка
Адреса – Кафедра теорії та технології програмування,
Київський національний університет ім. Тараса Шевченка,
01033, Київ, вул. Володимирська, 69,
E-mail: piter.protsyk@gmail.com

Композиційно-номінативний підхід до побудови семантики мови специфікації Z-Notation

Все більш очевидним стає те, що формальне моделювання програмних систем перетворюється на окрему дисципліну в галузі комп'ютерних наук. Проте вивчення даної тематики показує, що формальне моделювання поки що, на жаль, перебуває у стадії становлення, оскільки не існує єдиного послідовного викладу дисципліни та проблематики, достатньої чіткої класифікації методів, досліджуваних систем, та усталених понять. Більш того, для деяких широко розповсюджених методів навіть не існує чіткої, усталеної математичної семантики. Не завжди зрозуміло, який математичний апарат краще застосовувати при моделюванні тих чи інших типів систем.

У роботі пропонується підхід до побудови семантики для широко розповсюдженого методу Z на базі композиційно-номінативного підходу [1,2]. Z-Notation [3, 4, 5, 6] - це формальна мова специфікації програмних систем. Вона використовується для опису, моделювання та доведення властивостей функціональних аспектів послідовних систем.

Мова «Z-Notation» була розроблена колективом науковців Programming Research Group в Оксфордському університеті наприкінці 70 років. Заснована на математичній нотації, яка використовується в аксіоматичній теорії множин, лямбда-численні і логіці предикатів першого порядку. Існує стандартизований каталог (математичний інструментарій) часто вживаних функцій і предикатів. Всі вирази в Z-Notation є типізованими. Це дозволяє уникати деяких парадоксів наївної теорії множин.

Організацією стандартизації ISO (International Organization for Standardization) у 2002 році був розроблений стандарт Z-Notation. Він має назву – «Z Formal Specification Notation - Syntax, Type System and Semantics».

Під специфікацією програмної системи у Z-Notation розуміється набір схем, які описують її поведінку та властивості. Схема у Z-Notation – це засіб структурування специфікації. Схема складається з двох частин – декларативної та предикативної. У

декларативній частині проводиться опис змінних, а у предикативній на змінні накладаються обмеження. Існує три типи змінних – вхідні, вихідні та змінні стану системи. Синтаксично схема має такий вигляд: [Declaration | Predicates].

У специфікаціях схеми можна використовувати в наступних випадках:

- для завдання типів даних:

$$\text{Scheme} \cong [P : \text{TypeScheme}, \dots, | \text{Predicates}]$$

Наприклад:

$$\text{Month} == \text{jan} | \text{feb} | \text{mar} | \text{apr} | \text{may} | \text{jun} | \text{jul} | \text{aug} | \text{sep} | \text{oct} | \text{nov} | \text{dec}$$

$$\text{Date} == [\text{month} : \text{Month},$$

$$\text{day} : 1..31,$$

$$\text{year} : 1900..2100$$

$$|$$

$$\text{month} \in \{\text{sep}, \text{apr}, \text{jun}, \text{nov}\} \Rightarrow \text{day} \leq 30$$

$$\text{month} = \text{feb} \Rightarrow \text{day} \leq 29$$

$$]$$

Тут, схема з іменем Date задає множину календарних дат з певного проміжку часу. У предикативній частині накладаються обмеження на кількість днів у місяці.

- у визначеннях:

Схема: [Date | day = 31 • month] задає множину:

{jan, mar, may, jul, aug, oct, dec}, місяців у яких кількість днів дорівнює 31.

- у предикатах:

$$\forall \text{month} : \text{Month}; \text{day} : \mathbb{Z}; \text{year} : 1900..2100 \bullet \text{Date} \Rightarrow \text{day} \in 1..31$$

У загальному випадку схема має таку структуру:

$$\text{Scheme} = [x_1 : S_1 ; \dots ; x_n : S_n$$

$$x'_1 : S_1 ; \dots ; x'_n : S_n$$

$$i_1? : T_1 ; \dots ; i_m? : T_m$$

$$o_1! : U_1 ; \dots ; o_p! : U_p$$

$$|$$

$$\text{Pre}_k (i_1, \dots, i_m, x_1, \dots, x_n)$$

$$\text{Inv}_l (x_1, \dots, x_n)$$

$$\text{Inv}_o (x'_1, \dots, x'_n)$$

$$\text{Op}_j (i_1, \dots, i_m, x_1, \dots, x_n, x'_1, \dots, x'_n, o_1, \dots, o_p)$$

$$],$$

де

$x_1 : S_1 ; \dots ; x_n : S_n$ - змінні стану до виконання операції,

$x'_1 : S_1 ; \dots ; x'_n : S_n$ - змінні стану після виконання операції,

$i_1? : T_1 ; \dots ; i_m? : T_m$ - вхідні змінні,

$o_1! : U_1 ; \dots ; o_p! : U_p$ - вихідні змінні,

Pre_k - предикати, які задають передумови операції,

$\text{Inv}_1, \text{Inv}_o$ - інваріантні відношення, які накладаються на змінні,

Op_j - предикати, які задають операцію, тобто пов'язують поточний та наступний стан системи

Найбільш широко мова Z використовується для специфікації систем, заснованих на транзиційних моделях. В межах цієї моделі системи описуються в термінах станів та переходів з одного стану в інший. Функціонування системи починається з деякого початкового стану та продовжується (можливо, нескінченно довго) поки існує можливість переходу в інший стан. За допомогою схем можна описувати простір станів системи, початкові умови та переходи. Переходи задаються декларативно у формі передумова-постумова.

При такому тлумаченні системи можна досліджувати різні типи властивостей її моделі: циклювання, фінітність, детермінованість, тощо.

При використанні засобів мови Z існує можливість специфікувати такі аспекти моделей транзиційних систем:

- простір станів (множини станів) – StateSpace,
- початковий стан – InitialState,
- множина переходів – $\{\text{Op}_i \mid i \in I\}$.

Наприклад:

Простір станів:

$$\text{Counter} == [\text{value} : N \mid \text{value} < 100]$$

Початковий стан

$$\text{InitCounter} == [\text{Counter} \mid \text{value} = 0]$$

Операція

$$\text{Increment} == [\text{Counter} ; \text{Counter}' \mid \text{value}' = \text{value} + 1]$$

Кожна програмна система повинна мати хоча б один початковий стан. Тому, для кожної специфікації повинна виконуватись наступна формула (теорема в Z):

$$\exists \text{StateSpace} \bullet \text{InitialState}$$

Для наведеного прикладу:

$$\exists \text{value} \in N : \text{value} < 100 \wedge \text{value} = 0$$

Таким чином, мова Z має такі властивості:

- Стани – множина строго типізованих змінних (номінативних даних), задається схемами вигляду:

$$[v_1 : \text{Type}_1 , \dots , v_n : \text{Type}_n \mid \text{Constraint}_s(v_1 , \dots , v_n)]$$

$$\text{st} = \{ v_i \mapsto y_i \mid y_i \in \text{Type}_i \},$$

- Переходи – задаються схемами операцій:

$$[\text{st} , \text{st}' \mid \text{Pre}(\text{st}) , \text{Op}(\text{st}, \text{st}')],$$

- Мова опису процесів відсутня,
- Формули на станах задаються схемами вигляду: $[\text{st} \mid P(\text{st})]$,
- Транзиційні формули мають вигляд: $\text{Pre}(\text{st}) \rightarrow \text{op}(\text{st}, \text{st}')$, та задаються схемами переходів,
- Оператори для специфікації темпоральних властивостей відсутні.

Зауважимо, що хоча схеми є інтегральною частиною мови, їх семантика не достатньо повно вивчена [2]. Один з підходів до завдання семантики схем оперує поняттям підстановки (зв'язування). Підстановка 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 існує підстановка $z = \langle v_1 \Rightarrow u_1, \dots, v_n \Rightarrow u_n \rangle$ з компонентами $z.v_i$ рівними v_i відповідно. Тоді тип підстановки індукується таким чином: $\|z\| = \|v_1 : t_1, \dots, v_n : t_n\|$.

У роботі пропонується застосування композиційно-номінативного підходу для побудови семантики мови схем. Використання підходу дозволяє будувати семантику формальної мови на підставі єдиної методологічної платформи. У ній, стани системи описуються номінативними даними, переходи тлумачаться, як номінативні операції (операції над номінативними даними), які відображають один стан системи у інший. Операції будуються за допомогою чітко визначених композицій з базових відображень. Простір станів, множину початкових станів та операції визначають відповідні схеми: схеми станів, схеми ініціалізації та схеми операцій.

Побудова семантики мови Z схем повинна складатись з ряду етапів. Згідно принципу розвитку композиційно-номінативного підходу розглядається ієрархія мов від більш абстрактної до конкретної. На кожному рівні будується відповідна семантика.

Розпочнемо з побудови семантики спрощеної абстрактної мови AZ (Abstract Z). На відміну від мови Z , AZ – буде без типовою мовою. Усі змінні будуть приймати значення з деякого загального універсуму значень. Операції діють над усім станом системи. У визначенні схем відсутні вхідні та вихідні параметри, а також локальні змінні. Змінні поточного стану позначаються звичайним чином - x, y, z . Змінні стану після виконання операції позначаються символом ': x' , y' , z' '. Тоді, схема у мові AZ буде мати наступний загальний вигляд:

$SchemaText = StateSchemaText \mid InitSchemaText \mid OpSchemaText;$

На змінні стану обмеження не накладаються:

$StateSchemaText = [x_1, \dots, x_n];$

$InitSchemaText = [\mid Init(st)];$

$OpSchemaText = [\mid Pre(st); Op(st, st'); Post(st')];$

Специфікація системи у мові AZ буде складатись:

- із схеми $StateSchemaText$ – яка задає простір станів,
- схеми $InitSchemaText$ – яка визначає початковий стан,
- та довільної скінченної кількості схем $OpSchemaText$ – які задають переходи з одного стану в інший.

Кожна синтаксично правильна специфікація задає певну транзиційну систему, тобто модель програмної системи. Дослідження властивостей цієї моделі дає можливість виявляти помилки у специфікації на семантичному рівні.

Нехай V – множина імен, U – множина значень, D – множина номінативних даних, Op - множина всіх операцій $D \rightarrow D$.

Тоді, схема вигляду $StateSchemeText$ задає простір станів системи. Нехай $StateSchemeText = [x_1, \dots, x_n]$. Тоді, простір станів системи задається множиною номінативних даних, кожна з яких містить змінні з іменами x_1, \dots, x_n :

$SEM_{st} : SchemaText \rightarrow D.$

$SEM_{st}(StateSchemaText) = SEM_{st}([x_1, \dots, x_n]) = \{st \mid st \in D \wedge \{x_1, \dots, x_n\} \subseteq \eta[st]\} = StateSpace,$ де $\eta[st]$ – множина імен даного st .

Схема $InitSchemaText$ задає множину початкових станів системи:

$SEM_{st}(InitSchemaText) = SEM_{st}([\mid Init(st)]) = \{st \mid st \in StateSpace \wedge init(st)\} = INITIAL,$

де $\text{init}(st)$ – значення предикату Init на стані st .

Кожна схема вигляду OpSchemaText задає операцію переходу одного стану системи в інший:

$$\mathbf{SEM}_{\text{op}} : \text{SchemaText} \rightarrow (D \times D).$$

$$\mathbf{SEM}_{\text{op}}(\text{OpSchemaText}) = \mathbf{SEM}_{\text{op}}([\text{Pre}(st); \text{Op}(st, st'); \text{Post}(st')]) = \{(st, st') \mid st \in \text{StateSpace} \wedge st' \in \text{StateSpace} \wedge \text{pre}(st) \wedge \text{op}(st, st') \wedge \text{post}(st')\}$$

Задамо тепер семантику специфікації:

$$\mathbf{SEM}_{\text{TS}}(\text{SpecificationText}) =$$

$$\mathbf{SEM}_{\text{TS}}(\text{StateSchemaText}, \text{InitSchemaText}, \text{OpSchemaText}_1, \dots, \text{OpSchemaText}_k) =$$

$$(\mathbf{SEM}_{\text{st}}(\text{StateSchemaText}), st \in \mathbf{SEM}_{\text{st}}(\text{InitSchemaText}),$$

$$\{(st, st') \in \mathbf{SEM}_{\text{op}}(\text{OpSchemaText}_i) \mid i \in \{1, \dots, k\}\}) =$$

$$(\text{StateSpace}, P(st), \{\text{Op}_i(st, st') \mid i \in \{1, \dots, k\}\}) = \mathbf{TS}$$

Таким чином, за допомогою такого підходу по формальному опису специфікації програмної системи можна побудувати її транзиційну модель.

На наступному рівні будемо розглядати типізовану мову Typed AZ . Це мова більш конкретного рівня. В ній на значення кожної змінної будуть накладені додаткові обмеження. Обмеження будуть мати загальний вигляд: $x \in T$. Де x – це значення змінної з іменем x , а T – деяка множина. Можна вважати, що T – це носій деякого типу з іменем T .

Синтаксичні зміни у мові не досить суттєві:

$$\text{StateSchemaText} = [x_1 : T_1, \dots, x_n : T_n];$$

На семантичному рівні зміни будуть більш суттєвими. Вони накладуть обмеження на простір станів системи:

$$\mathbf{SEM}_{\text{st}} : \text{SchemaText} \rightarrow D.$$

$$\mathbf{SEM}_{\text{st}}(\text{StateSchemaText}) = \mathbf{SEM}_{\text{st}}([x_1 : T_1, \dots, x_n : T_n]) = \{st \mid st \in D \wedge \{x_1, \dots, x_n\} \subseteq \eta[st] \wedge \forall i \in \{1..n\} : st(x_i) \in T_i\} = \mathbf{StateSpace}.$$

Обмежений таким чином простір станів системи буде мати ряд переваг важливих властивостей. Головною з цих властивостей буде можливість здійснювати безпосередній контроль за значеннями змінних у стані системи.

У більш загальному випадку, маємо:

$$\mathbf{SEM}_{\text{st}} : \text{SchemaText} \rightarrow D.$$

$$\mathbf{SEM}_{\text{st}}(\text{SchSp}) = \mathbf{SEM}_{\text{st}}([\text{Declaration} \mid \text{Predicates}]) = \mathbf{SEM}_{\text{st}}([x_1 : S_1; \dots; x_n : S_n \mid P_i(\bar{x}), i \in I]) = \{[x_1 \mapsto u_1, \dots, x_n \mapsto u_n] \mid u_i \in S_i, i \in \{1..n\}, P_j(u_1, \dots, u_n) = \text{True}, j \in I\} \Leftrightarrow$$

$$\{ [x_1 \mapsto u_1, \dots, x_n \mapsto u_n] \mid u_i \in S_i, i \in \{1..n\}, P_1(u_1, \dots, u_n) \wedge \dots \wedge P_k(u_1, \dots, u_n), I = \{1, \dots, k\} \}.$$

Схеми вигляду $[x_1 : S_1 ; \dots ; x_n : S_n \mid P_i(\bar{x}), i \in I]$ надалі будемо називати схемами-визначеннями. Схеми-визначення у композиційно-номінативній семантиці визначають номінативну множину. Вони використовуються для опису простору станів та складних типів даних.

Перейдемо до розгляду наступного типу схем:

$$\mathbf{SEM}_{ini} : \text{SchemaText} \rightarrow D.$$

$$\mathbf{SEM}_{ini}(\text{SchIni}) = \mathbf{SEM}_{ini}([\text{Declaration} \mid \text{Predicates}]) = \mathbf{SEM}_{ini}([x_1 : S_1 ; \dots ; x_n : S_n \mid P_i(x_1, \dots, x_n), i \in I]) = \{ \text{st} \mid \text{st} \in \Gamma \wedge P_i(\text{st}(x_1), \dots, \text{st}(x_n)) \}, i \in I \}.$$

Схеми вигляду $([x_1 : S_1 ; \dots ; x_n : S_n \mid P_i(x_1, \dots, x_n), i \in I])$ називаються схемами-ініціалізаторами. Схеми-ініціалізатори задають множину початкових станів системи.

Залишається ще один тип схем. Це схеми-операції вигляду:

$$([x_1 : S_1 ; \dots ; x_n : S_n ; x_1' : S_1 ; \dots ; x_n' : S_n \mid P_i(x_1, \dots, x_n, x_1', \dots, x_n'), i \in I]).$$

Схеми операції визначають множину станів у які може перейти система з поточного стану наступним чином:

$$\mathbf{SEM}_{op} : \text{SchemaText} \rightarrow (D \rightarrow D).$$

$$\begin{aligned} \mathbf{SEM}_{op}(\text{SchOper}) &= \mathbf{SEM}_{op}([\text{Declaration} \mid \text{Predicates}]) = \mathbf{SEM}_{op}([x_1 : S_1 ; \dots ; x_n : S_n ; x_1' : S_1 ; \dots ; \\ &x_n' : S_n \mid P_i(x_1, \dots, x_n, x_1', \dots, x_n'), i \in I]) = \{ P_1(x_1, \dots, x_n, x_1', \dots, x_n') \wedge \dots \wedge P_k(x_1, \dots, x_n, x_1', \dots, \\ &x_n') \} = \\ &= Q(x_1, \dots, x_n, x_1', \dots, x_n') = Q_1(x_1, \dots, x_n) \wedge Q_2(x_1, \dots, x_n, x_1', \dots, x_n') \wedge Q_3(x_1', \dots, x_n') \mid = \\ &= \{ s \mid s \in \text{st}' \wedge Q(\text{st}(x_1), \dots, \text{st}(x_n), s(x_1), \dots, s(x_n)) \} = \mathbf{Op}(\text{st}). \end{aligned}$$

У загальному випадку семантика схем задається наступним чином:

$$\mathbf{SEM}_{sh}(\text{Schema}) =$$

$$\left\{ \begin{array}{l} \mathbf{SEM}_{st}(\text{Schema}), \text{ якщо Schema} \\ \text{має вигляд } [x_1 : S_1 ; \dots ; x_n : S_n \mid P_i(x_1, \dots, x_n), i \in I], \\ \mathbf{SEM}_{ini}(\text{Schema}), \text{ якщо Schema} \\ \text{має вигляд } [x_1 : S_1 ; \dots ; x_n : S_n \mid P_i(x_1, \dots, x_n), i \in I] \\ \mathbf{SEM}_{op}(\text{Schema}), \text{ якщо Schema} \\ \text{має вигляд } [x_1 : S_1 ; \dots ; x_n : S_n, x_1' : S_1 ; \dots ; x_n' : S_n \mid P_i(x_1, \dots, x_n, x_1', \dots, x_n'), i \in I] \end{array} \right.$$

Специфікація системи складається з чотирьох типів об'єктів – схем-станів, схем-операцій, схем-ініціалізаторів та аксіоматичних визначень. Аксіоматичні визначення подібні до схем станів – вони вводять нові змінні та накладають на них обмеження. Значення цих змінних не можуть змінюватись в процесі роботи системи. Аксіоматичні визначення задають константи.

Як уже було сказано специфікація системи складається з набору параграфів Z, параграф – це або декларування нового типу, або аксіоматичне визначення, або схема, або коментарі. Коментарі не мають формального семантичного значення, тому вони не будуть прийматись до уваги.

Після створення специфікації та побудови математичної моделі системи. Постає задача її аналізу та доведення властивостей. Властивості формуються у вигляді теорем деякої аксіоматичної теорії. У класичному варіанті – це теорія множин Цермело-Френкеля. Далі доведення відбувається у рамках цієї теорії, але такий підхід слабо враховує специфіку предметної області. Проблеми, які при цьому виникають аналізуються в роботі [7].

У наступних роботах буде проводитись дослідження властивостей предикатів, що задають операції, побудові та аналізі відповідних логік. Введення додаткових класів змінних, зокрема локальних визначень. Дослідженні семантичних зв'язків між різними рівнями мови, розробленні засобів доведення властивостей Z-специфікацій.

ЛІТЕРАТУРА:

1. Никитченко Н.С. Композиционно-номинативный подход к уточнению понятия программы: Проблемы программирования.-1999.-№1, стр. 16-31.
2. Редько В.Н. Основания композиционного программирования: Программирование. – 1973.-№3-стр. 3-13.
3. Jim Woodcock, Jim Davies: Prentice Hall - Using Z. Specification, Refinement, and Proof, 523 p, 1996.
4. Brien S.M., Martin A.P.: Academic Press – Symbolic Computation volume 11: A Calculus for Schemas in Z, pages 1-29, 1999.
5. J.M. Spivey: Prentice Hall Education - The Z Notation: A Reference Manual.-1992.
6. Jonathan Bowen: On-line book – Formal Specification and Documentation using Z: A case study approach.-2003.
7. Andrew Martin: FM'99, Volume II, LNCS 1709 - Relating Z and First-Order Logic. – 1999.