

П.П. Процик

ВЕРИФІКАЦІЯ ПРОГРАМ ЗАСНОВАНА НА АВТОМАТИЗОВАНІЙ ПОБУДОВІ ТЕСТОВИХ ПРИКЛАДІВ

(Київський Національний Університет ім. Т. Г. Шевченка, факультет кібернетики)

З початку створення першого обчислювального пристрою і до сьогоднішніх днів актуальною проблемою є створення надійних, якісних та коректних програмних систем. Цілком зрозуміло, що розв'язання такої комплексної проблеми не можливе за рахунок просування тільки одного з аспектів комп'ютерних наук. На даний момент розроблена ціла низка методів та засобів розв'язання окремих питань з цієї області. У даній роботі представлено прагматико обумовлений підхід до верифікації програм, що базується на автоматизованій побудові тестових прикладів. Він охоплює процеси специфікації програм, перетворення специфікацій у програму, що виконується, подальшу її верифікацію та перевірку відповідності поведінки програми початковій специфікації. Ця методика ґрунтується на науково-методологічному базисі композиційно-номінативного підходу [1] та є подальшим розвитком ідей представлених у роботах [2,3,4].

В якості мови специфікації застосовується формальна мова *Z-Notation* [5]. Її семантика тлумачиться у розумінні, представленому в роботі [3]. Найбільш широко мова *Z* використовується для специфікації систем, заснованих на транзиторних моделях. У рамках цієї моделі, програмна система описується в термінах станів та переходів з одного стану в інший.

Основною синтаксичною конструкцією та одиницею специфікації мови *Z* є схема. Синтаксично вона складається з двох частин: декларативної – у якій описуються змінні та предикативної – у якій на змінні накладаються обмеження у вигляді предикатів: *Схема* = [*Декларації* | *Предикати*]. Якщо у декларативній частині змінна записана з символом (*'*), то це означає, що значення змінної зміниться після застосування схеми. В залежності від типу змінних та предикатів будемо розглядати такі три класи схем:

1. Схеми, що задають простір допустимих станів – *StateSchemaText* = [x_1, \dots, x_n | $P(x_1, \dots, x_n)$];
2. Схеми, що задають початкові стани – *InitSchemaText* = [x_1', \dots, x_n' | $Init(x_1', \dots, x_n')$];
3. Схеми, що задають операції – *OpSchemaText* = [$x_1, \dots, x_n, x_1', \dots, x_n'$ | $Pre(x_1, \dots, x_n); Op(x_1, \dots, x_n, x_1', \dots, x_n'); Post(x_1', \dots, x_n')$];

Специфікацією програмної системи у *Z-Notation* будемо називати четвірку: *Spec*=(*ST, I, OP, C*), де *ST*={*ST*₁, ..., *ST*_{*k*}} – множина схем станів, *I*={*I*₁, ..., *I*_{*m*}} – множина схем, які задають початковий стан, *OP*={*Op*₁, ..., *Op*_{*n*}} – множина схем операцій, *C*={*C*₁, ..., *C*_{*p*}} – множина допоміжних схем.

Кожна специфікація у такій формі може бути перетворена у програму на мові *CNLS* [4]. 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.

Розглянемо типовий приклад схеми, що задає операцію, та функцію яка може бути отримана за нею [4]:

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

Функція: *function* *f*(*x*) [*pre*($x > 0$); *post*($x \% 2 == 0$); *invariant*(*x is int*);] { *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. Як приклад, розглянемо таке уточнення функції *f*:

function *f*(*x*) [*pre*($x > 0$); *post*($x \% 2 == 0$); *invariant*(*x is int*);] { $x = x * 4$; *return* *x*; }.

Процес верифікації на основі тестових прикладів є ітеративною процедурою для деякої функції визначеної у програмі або для програми в цілому. При цьому будемо використовувати систему *Yices* [6] для відшукання прикладів, що задовольняють умови контракту функції.

Система *Yices* відноситься до класу *SMT*-солверів та використовується для перевірки на суперечність теорій першого порядку. Крім того, її можна використовувати для відшукання прикладів, які задовольняють умовам деякої заданої теорії, або доводять її суперечність.

Метод верифікації складається з таких кроків:

1. Використовуючи умови контракту, побудувати набір даних – *d*₁, що буде його задовольняти;
2. Отриманий набір даних підставити на вхід функції, та обчислити її значення – *y*₁. Якщо значення функції невизначене (функція заиклилась, виникла виключна ситуація, тощо), тоді вважаємо, що програма обчислення функції містить помилку – необхідно або посилити умови у специфікації, або виправити помилку та перейти до кроку 1;
3. До умов, що були побудовані на першому кроці, додаємо нове твердження: *f*(*d*₁) = *y*₁. Та перевіряємо, що отримана система тверджень не містить суперечностей; Якщо суперечність існує – модифікуємо код програми або її специфікації.

- Далі продовжуємо процедуру пошуку нових тестових даних – додаючи на кожному кроці умову $x \neq x_t$ або (*assert* ($\neq x x_t$)), де x_t – значення змінної отримане на попередньому кроці. Та переходимо до кроку 2.

Якщо на деякому кроці процедура призведе до негативного результату на етапі перевірки значення функції – це одразу означає, що програма, яка обчислює значення функції не відповідає умовам специфікації, а отже містить помилку.

Розглянемо, як приклад, декілька кроків цієї процедури з використанням описаних засобів автоматизації. Спочатку перетворимо умови контракту функції f з наведеного прикладу в предикати записані на мові обраного солвера. Ця процедура є цілком технічною і підлягає автоматизації. Дістанемо такі співвідношення:

Контракт CNLS	Yices
<i>invariant</i> (x is int)	(<i>define</i> $f1::(-> int int)$)
<i>pre</i> ($x > 0$);	(<i>define</i> $x::int$)
<i>post</i> ($x \% 2 == 0$);	(<i>assert</i> ($> x 0$))
	(<i>assert</i> ($= (mod (f1 x) 2) 0$))

Після виконання *Yices* отримаємо, що умови задовольняються, якщо $x=1$ та $f(x) = 0$. Цей, результат впливає лише на основі умов контракту, тому значення функції $f(x) = 0$ може відрізнитись від реального значення функції на цьому даному. Поки що, це не є важливим, оскільки мета першого кроку – отримати вхідні дані, які б задовольнили умови контракту. На другому кроці – виконується обчислення функції $f(x)$ при $x=1$. Дістанемо такий результат $f([x \rightarrow 1]) = x \Rightarrow *4 = 4$. Далі процедура продовжується з додаванням нових умов: $x=1$ та $f(x) = 4$. При чому, можливі два випадки – система умов містить суперечність, і значить програма не відповідає специфікації, або суперечності немає і можна продовжувати процес далі. В нашому прикладі отримане в результаті обчислень функції значення (стан системи) задовольняє умовам контракту. А отже, функція на цьому конкретному даному задовольняє умови початкової специфікації.

За рахунок автоматизації, цю процедуру можна використовувати для перевірки відповідності реалізації початкової специфікації на великих об'ємах даних, і тим самим суттєво зменшуючи ризик появи дефектів у кодї. При цьому, процес не важко адаптувати для інших мов специфікації та програмування.

Більш того, комбінуючи цей емпіричний підхід з математичними методами можна одержати і більш сильні твердження про властивості досліджуваної програми.

Висновки

Розроблено та представлено метод верифікації програм на основі побудови тестових прикладів. Продемонстровано його застосування для випадку мови специфікації *Z-Notation* та спеціальної мови програмування *CNLS*. Продемонстровано принципову можливість автоматизації розробленої методики, що є важливим для розв'язання задач такого класу.

Література

- Никитченко Н.С. Композиционно-номинативный подход к уточнению понятия программы. // Проблемы программирования.– 1999.–№1. С. 16– 31.
- П.П. Процик, Run-time верифікація програм у композиційно-номінативній мові програмування Script.NET // Тези доповідей Міжнародної конференції «Теоретичні та прикладні аспекти побудови програмних систем» (TAAPSD'2008). Київ. – 2008, 22-24 вересня 2008, 110-117 с.
- Процик П.П. Композиційно-номінативний підхід до побудови семантики *Z-Notation* // Вісник київського університету. Сер.: Фізико-математичні науки. – 2008. – №1. –С. 116–120.
- Процик П.П. Методика автоматизованої трансформації формальних специфікацій у програмний код, що виконується // Матеріали XIV науково-технічної конференції студентів, магістрантів, аспірантів та викладачів, Запорізька Державна Інженерна Академія. Запоріжжя, 13-17 квітня 2009 року, 153-154 с.
- Jim Woodcock, Jim Davies: Using Z. Specification, Refinement, and Proof –New York: Prentice Hall, 1996. – 523 p.
- B. Dutertre, L. de Moura. The YICES SMT Solver // Computer Science Laboratory. SRI International. <http://yices.csl.sri.com/tool-paper.pdf>, (July, 2009).