УДК 004.415.2.043

САЛАПАТОВ В.І.

МОДЕЛЮВАННЯ, ВЕРИФІКАЦІЯ ТА РОЗРОБКА ПРОГРАМ

У статті розглядається удосконалення методу MODEL CHECKING при створенні програм. При цьому технічне завдання у вигляді низки вимог, які записуються за допомогою рівнянь темпоральної логіки, використовуються безпосередньо для побудови автоматної моделі майбутньої програми. Таким чином, при створенні моделі одразу здійснюється її верифікація, оскільки вимоги будуть виконуватись у процесі побудови моделі. В подальшому автоматна модель, яка являє собою логічну структуру програми, використовується для створення програми на будь-якій процедурній мові програмування.

In the paper the improvement of the method MODEL CHECKING in programming is considered. This technical problem consist of a number of requirements that are represented using temporal logic equations and used directly to build the automaton model for future programs. Therefore, when creating models at once by its verification, since the requirements will be implemented in the model creating. Further automaton model that represents the logical structure of the program used to create applications on any procedural programming language.

Вступ

Коректність роботи програми, яка підлягає розробці, формально може бути доведена тільки по відношенню до її моделі. При цьому модель алгоритму майбутньої програми або її семантика являють собою спрощену абстракцію, яка може бути визначена за допомогою певних специфікацій у вигляді логічних обмежень або формул темпоральної логіки [1]. Перехід з одного стану автоматної моделі визначається не тільки змінами внутрішніх змінних, але і зовнішніх змінних (типа volatile). В такому разі стани моделі програми умовно розділені на такі, що залежать від зміни власних змінних, та ті, які змінюються ззовні. Час зміни таких змінних є не передбачуваним і повязаний із зовнішніми перериваннями. Тому перехід у стани, які викликані такими змінами, складають окремі випадки і пов'язані із синхронізацією паралельних процесів. Переходи у такі стани та порядок їх обробки мають описуватись окремо. Коректність роботи будь-якої паралельної програми залежить від коректності опису таких станів моделі. Тому для перевірки роботи паралельних програм особливо важливим є контроль поведінки моделі цих програм саме в таких станах, що викликаються зовнішніми подіями.

Суперпозиція графів моделі програм

Для створення комплексної моделі складних програм доцільно створити моделі усіх незалежних програмних модулів або гілок. Кожна з таких гілок має бути визначена сукупністю обмежень та специфікацій у вигляді рівнянь темпоральної логіки S. Після побудови графів автоматних моделей та їхньої верифікації треба виконати суперпозицію цих моделей і створити спільну модель. Під час такої сборки моделі - $Sc = \bigcup_{i=1}^{N} Si$ можуть з'явитись додаткові специфікації, тобто модель буде уточнюватись. Тут Sc об'єднання специфікацій від кожної незалежної гілки паралельної програми. Крім того, така сборка може виявити однакові специфікації у вигляді рівнянь в різних гілках паралельної програми. А це призводить до спрощення моделі програми - спільні специфікації можуть зменшити кількість станів моделі паралельної програми та породження нових станів.

Такий процес нормалізації графової моделі програми дозволяє оптимізувати її за кількістю станів та переходів між цими станами. Схожі вимоги специфікацій, якім мають відповідати окремі програмні фрагменти, можуть бути визначені як частина дій у кожному фрагменті моделі, або як спільна частина її нового стану. В першому випадку спільній частині видповідатиме певна підпрограма, а в другому - нова об'єднана вершина на графовій моделі. В такому випадку процес верифікації розбивається на кілька етапів і значно спрощується. В разі оформлення підпрограм специфікації, які визначають відповідні частини програмної моделі, мають повністю співпадати і знаходитись в різних частинах графа. Якщо вони сходяться до одного стану, то тоді вони створюють новий спільний стан моделі програми. При цьому повне співпадіння специфікацій не обов'язкове, їхні відмінності можуть породжувати певні розгалуження. Таким чином, процес верифікації мо-

делі програми на відміну від класичної технології MODEL CHECKING спрощується і не вимагає наявності ліцензійних і коштовних програм-верифікаторів. Такий процес удосконалення і спрощення програмних моделей нагадує нормалізацію баз даних, коли данні, з одного боку стиснюються, а з іншого - зберігається їхня цілісність і несуперечливість. В процесі такого спрощення спочатку визначаються схожі специфікації моделі, а потім здійснюється спрощення моделі.

Побудова програми за її моделлю

Після того, як в результаті побудови та нормалізації автоматної моделі програми виконана її перевірка на відповідність заданим специфікаціям можна перейти безпосередньо до створення програми. Обрання мови програмування в якості процедурної, інструментальної мови залежить від вимог замовника та міркувань виконавця. Власно процес програмування полягає у послідовному обході графу автоматної моделі, починаючи з її початкового стану до кінцевого стану по всіх можливих шляхах. Лінійні дії в кожному стані моделі автомату реалізуються відповідними мовними конструкціями, а функції переходу - за допомогою розгалужень. Такий підхід створення програм дозволяє перейти від інтуітивної побудови блок-схем алгоритмів програм до формалізації побудови їх автоматних моделей для наступного безпосереднього кодування. Логічна структура програми повинна створюватись згідно вимог технічного завдання на розробку. Треба також зпзначити, під час моделювання можуть виявитись недоліки і навіть помилки в специфікаціях технічного завдання, які легше і простіше виявити саме на цьому етапі. Крім того, на цьому етапі простіше вносити доповнення і можливі зміни в технічне завдання у вигляді нових специфікацій у модель.

Автоматна модель програми

Кінцеві автомати є дуже зручним формалізмом, який кінцевим чином задає звичайні формальні мови - нескінченні множини ланцюжків кінцевої довжини. Основою для побудови автоматної моделі програми є структура Крипке [1] Модель програми у вигляді кінцевого автомату створюється згідно її логічного опису та математичної моделі. Особливістю побудови такої моделі є наявність у кожному її стані власної функції збудження, що ускладнює подання автоматної моделі у табличному вигляді. Функції переходу з одного стану в інший також визначаються в кожному стані по-різному. Побудову автоматної моделі покажемо на прикладі створення програми обробки виразу за стековим алгоритмом [2]. Для опису математичної моделі будемо користуватись апаратом темпоральної логіки [1]. Тоді для обробки виразу його синтаксис та семантику можна описати наступним чином.

get (1); t U r (2);
$(p(t_i) > (t_{i-1}))$ U tostack (3);
$(p(t_i) \geq p(t_{i-1}))$ U code, fromstack (4);
(stack = Ø V (r = fin) U END (5);
(stack = Ø V (r ≠ fin) U get) (6);
(t_b = '-' V r = Ø) U (r = 0) (7);
(t = '(') V r ≠ Ø) U funct (8);
(t = '(') V r = Ø) U tostack (9);
(t = '(') V r = Ø) U Ee t = ')' (10);
(t_{i+1} = ')') V r ≠ Ø V t_i ≠'(') U goto 4 (11);
(t_i = '(' V t_{i+1} = ')') U lost, razd (12).

Робота алгоритму починається з пошуку пар лексем t та r – лексеми даних та лексеми дії (1). Формула (2) вказує на те, що елементи виразу складаються з пар лексем: лексеми даних - r та лексеми дії - t. Друга формула (3) вказує, якщо пріоритет лексеми дії у поточній парі лексем більше за такий самий пріоритет у попередній парі, то поточна пара лексем зберігається у стеку. Наступна формула вказує (4), що у звортному випадку породжується проміжний код та перевіряється стек. Формула 5 вказує, що програма завершується, коли стек пустий та лексема дії вказує на кінець виразу. Якщо лексема дії не вказує на кінець виразу (6) то продовжується пошук чергової пари лексем. Формула 7 описує варіант наявності унарного мінусу на початку виразу. У формулах 8-12 враховуються особливості обробки дужок. Так, формула 8 визначає обробку функції (після ідентифікатора – ліва дужка). Формула 9 визначає обробку лівої дужки без лексеми даних як високо пріоритетну операцію. Формула 10 вказує, що коли є права дужка, то до кінця виразу (Ee) обов'язково повинна з'явитись права дужка. У формулі 11 визначається порядок дій при обробці правої дужки, коли попередня лексема дії не є правою дужкою, тобто перехід до формули 4. Якщо стек виявиться пустим, то маємо помилку порушення парності дужок. І, нарешті, остання формула (12) вказує на умову знищення дужок (lost) та пошук після правої дужки лексеми дії (razd). Треба зазначити, що правила включають використання певних функцій. Функція get означає пошук пари лексем даних та дії. Функція tostack передбачає запис у стек пари лексем

даних та дії. При виконанні функції fromstack аналізується стек та якщо він не пустий, то виконуються формула 4, інакше - формули 5 та 6. Функція code вказує на породження коду, а функція lost на знищення парних дужок. Функція END вказує на умову закінчення роботи алгоритму обробки виразу. Для того, щоб побудувати автоматну модель алгоритму, треба визначити порядок застосування формул. Частково це вже було зроблено стосовно функції fromstack

Оскільки модель програми створюється у процесі її розробки і при цьому ще й перевіряється, то верифікація її моделі вже не потрібна. Коректність створеної програми визначається коректністю її автоматної моделі згідно специфікацій.

Побудова автоматної моделі програми та її верифікація

Формули темпоральної логіки більш конкретно дозволяють і точно описати модель програми ніж більш абстрактні та менш інформативні форми Бекуса-Наура. Для створення автоматної моделі треба пов'язати порядок застосування цих формул. Коли це буде зроблено, фактично отримаємо автоматну модель програми. Початковий та кінцевий стани автоматної моделі визначені у формулах 1 та 5. Отже, в разі наявності пари лексем (формули 1 та 2) наступним є застосування формул 3 та 4. Якщо лексема даних відсутня, то застосовуються формули 7, 9, 10. Формула 8 передбачає обробку функцій, які сприймаються як лексеми даних та обробляються окремою процедурою. Формули 3 та 4 застосовуються за наявності двох пар лексем, причому формула 5 визначає кінцевий стан автоматної моделі.

Формула 7 може бути застосована спільно з формулами 7, 9, 10, тобто функція переходу визначається умовами в цих формулах і програмно реалізується у вигляді розгалуження. Графічно модель програми представлена на малюнку нижче.

Стан 4 має додаткові стани, які ілюструють виконання функцій формування коду та аналізу стеку - code та fromstack. Стан 12 також розбитий на два стани, щоб показати наочно процедури lost та razd. Виділення станів 4_1 та 4_2 дозволило оптимізувати граф і виконати перехід із стану 11 у стан 4_1, а спільне використання процедури tostack у станах 2 та 9 – використати однакові дії у стані 3.

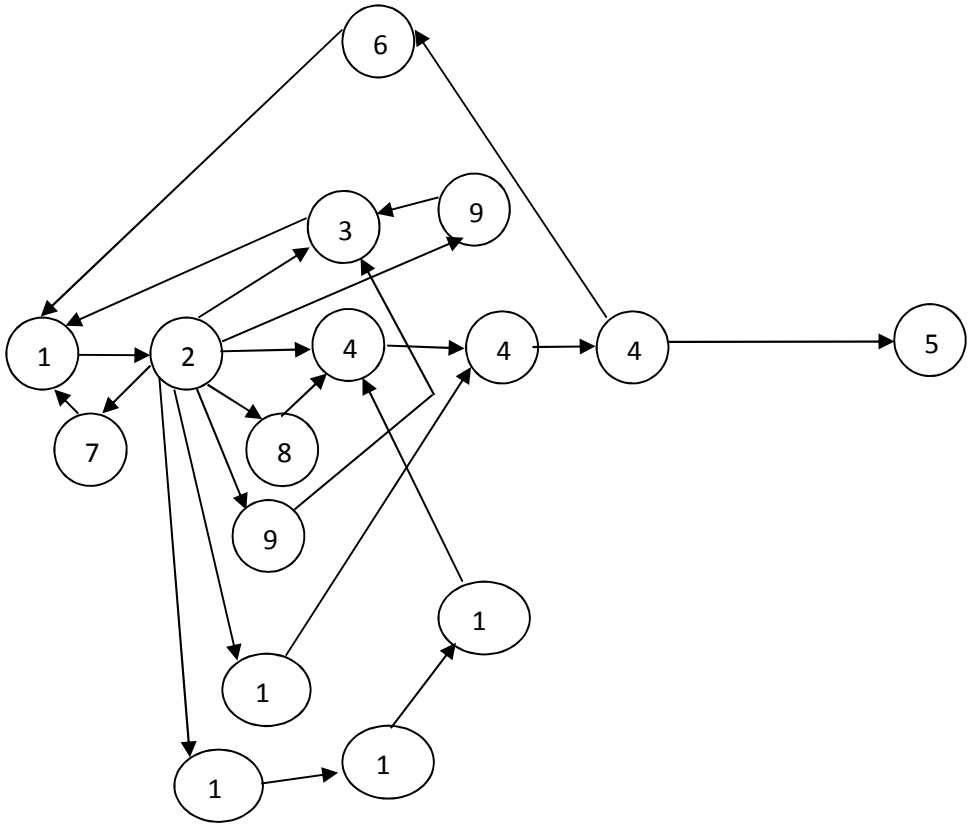

Рис.1. Граф моделі програми обробки виразу

Висновки

На даному прикладі показана формалізація опису програм в термінах темпоральної логіки, частково застосована суперпозиція окремих станів. При побудові графічної моделі це дозволило уточнити точки переходу з одного стану в інший і таким чином оптимізувати графову модель. Під час опису моделі перевірялось виконання формул темпоральної логіки, а також здійснювались зв'язки при переході від одних формул до інших. Тому процес верифікації моделі програм здійснювався безпосередньо при її створенні як обов'язковий етап технологічного процесу. При створенні більш складних програмно-апаратних систем доцільно виконати розбиття їх на окремі складові, побудувати їхні автоматні моделі, а потім виконати комплексну верифікацію цих моделей та здійснити остаточну перевірку. Такий підхід усуває застосування програм-верифікаторів, дозволяє перевірити виконання формул темпоральної логіки і спрощує процес створення складних систем. А розширення темпоральної логіки додатковими вказівками щодо переходу з одного стану в інший додатково полегшують побудову моделі.

Список посилань

1. Карпов Ю.Г. MODEL CHECKING. Верификация параллельных и распределенных программных систем. /Ю.Г. Карпов – СПб.; БХВ-Петербург, 2010.- 560 с.
2. Салапатов В.І. Синтаксичний аналіз із розподілом лексем на групи / В. Салапатов// Вісник національного технічного університету України «КПІ», Інформатика, управління та обчислювальна техніка. Київ. – 2008. - № 49. - С. 29-33.