https://orcid.org/0000-0001-7567-637X

УДК 519.17:004.942

В.І. САЛАПАТОВ*

ПОРЯДОК ОПИСУ І ОБРОБКИ ГРАФА АВТОМАТНОЇ МОДЕЛІ ПРОГРАМИ

* Черкаський національний університет ім. Б. Хмельницького, м. Черкаси, Україна

Анотація. *У статті розглядається метод опису моделювання програм за допомогою предикатів, у результаті якого створюється модель у вигляді недетермінованого скінченного автомата. Пропонується формат опису предикатів із розділенням змістовної та умовною частин. Для опису змістовної частини предиката мають застосовуватись арифметичні оператори, дужки, стандартні функції та оператори темпоральної логіки U та N. Для логічної частини предикатів, крім логічних функцій, пропонується застосовувати також операції відношення і арифметичні оператори, дужки, стандартні функції. Формування опису кожного стану автоматної моделі програми має завершуватись розгалуженням по різних гілках моделі. Для опису моделей паралельних програм введені спеціальні стани: стан-монітор для доступу різних процесів до спільних ресурсів та стан-протокол для опису незалежних паралельних гілок. Модель створюється у процесі її опису, тому не потребує подальшої верифікації. Якщо опис моделі виконаний коректно і при цьому обраний оптимальний алгоритм майбутньої програми, то така модель повністю відповідатиме її опису. При цьому відпадає необхідність у верифікації моделі на відміну технології MODEL CHECKING, яка вимагає верифікацію. Граф отриманої моделі обробляється шляхом її послідовного обходу по всіх гілках з поверненнями у попередні стани та наступною програмною реалізацією. Послідовний обхід має виконуватись для кожної гілки моделі або до кінцевого її стану, або до стану, який вже був оброблений при обході. Обробка моделі полягає у трансляції опису моделі у внутрішнє подання для наступного перетворення у програму на цільовій процедурній мові програмування. При цьому усі дії у змістовних частинах предикатів, а також умови розгалужень у процесі трансляції перетворюються у внутрішнє подання програми. Дана технологія забезпечує пряме перетворення опису моделі програми у саму програму.*

Ключові слова: *предикати, недетермінований скінченний автомат, граф моделі, обхід графа моделі.*

Abstract. *The article considers the method for describing program modeling where predicates are used to create a model in the form of a nondeterministic finite automaton. A format of the description of predicates with the division of their semantic and conditional parts is offered in the paper. Arithmetic operators, parentheses, standard functions, as well as temporal logic operators U and N should be used to describe the content of the predicate. For the logical part of predicates, in addition to their logical functions, it is proposed to use relation operations, arithmetic operators, parentheses and standard functions. The formation of the description of each state of the program automaton model should be completed by dividing into different branches of the model. To describe the models of parallel programs, the following special states have been introduced: a state-monitor for the access of various processes to shared resources and a state-protocol for the description of independent parallel branches. The model is created in the process of its description, so it does not require further verification. If the description of the model is performed correctly and the optimal algorithm of the future program is selected, such model will fully correspond to its description. Unlike MODEL CHECKING technology, which requires verification, it eliminates the need for model verification. The graph of the obtained model is processed by its sequential traversal on all branches with returns to the previous states and subsequent software implementation. Consecutive traversing should be performed for each branch of the model either to its final state or to the state that has already been processed during the traversal. Model processing includes transmission of the model description into an internal representation for subsequent conversion into a program in the target procedural programming language. In this case all actions in the substantive parts of the predicates, as well as the conditions of the branches in the process of transmission are converted into an internal representation of the program. This technology provides a direct conversion of the description of the program model into the program itself.*

Keywords: *predicates, nondeterministic finite automaton, model graph, model graph bypass.*

DOI: 10.34121/1028-9763-2021-3-121-125

1. Вступ

Побудова коректних і надійних програм досі ще залишається актуальною і нагальною проблемою. Особливо це важливо для програм управління. Через помилки у програмах управління зазнали катастроф літаки BOING 737 в Індонезії (жовтень 2018 р.) і Ефіопії (березень 2019 р.). А через помилку у програмі управління деяких моделей авіалайнерів Airbus A350, як і раніше, необхідно перезавантажувати систему кожні 149 годин, щоб запобігти часткову або повну втрату функціональності окремих систем [1]. Тому проблема коректності і надійності програм досі ще залишається актуальною.

Як вже зазначалося у [2], опис моделі програми можна здійснити за допомогою предикатів. Сучасним методом моделювання та розробки програм є подання моделі у вигляді недетермінованого скінченного автомата. При цьому в разі коректного опису моделі сама модель вважається коректною і не потребує додаткової верифікації. Це відбувається через те, що модель створюється під час її опису. Змістовна частина предиката описує функцію виходу у певний стан, в якому мають виконуватись певні дії, тобто функцію виходу. А логічна його частина описує умову виконання змістовної частини.

Для опису функції виходу мають застосовуватись звичайні арифметичні операції, дужки, стандартні функції. Крім того, можуть бути використані темпоральні оператори p U q та N r [3]. Оператор p U q забезпечує виконання предиката p, якщо він істинний, до тих пір, поки істинним не стане предикат q. Оператор N r має застосовуватись для опису паралельних гілок алгоритму програми та розгалужень.

Метою даної статті є спроба створення програм за допомогою опису їх моделей, тобто мінімізації власно процесу програмування.

2. Методика опису моделей та їх обробка

Для опису функції переходу мають використовуватись логічні функції, операції відношення (>, <, >=, <=, =, ≠) а також темпоральні оператори p U q та N r [3]. Таким чином, предикати описують автоматну модель з її станами і функціями переходу. Для виділення логічної частини предиката можна застосувати фігурні дужки, а для виконавчої частини – квадратні дужки. Тоді предикат має такий вигляд {умова} [перелік дій].

Для внутрішнього подання автомата зручним є застосування простої бази даних із таких двох відношень, як показано на рис. 1.

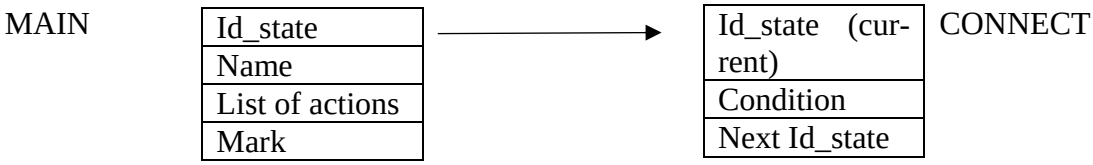

Рисунок 1 – Структура бази даних моделі програми

Перше відношення MAIN описує всі стани моделі з функціями дії. Атрибут Id_state являє собою ідентифікатор стану, атрибут Name вказує на ім'я стану. Для опису дій у даному стані застосовується атрибут List of actions типу memo, в якому мають бути описані всі дії у даному стані. Атрибут Mark логічного типу повинен вказувати на те, що при подальшій обробці бази даних до даного стану вже було звернення. Друге відношення CONNECT пов'язує всі стани між собою. Відношення пов'язані через ідентифікатор стану Id_state. Атрибут Condition вказує на перехід в інший стан через ідентифікатор наступного стану Next Id_state. Атрибут Condition має тип memo і вказує на умови розгалуження. Тип memo являє собою символьний тип невизначеної довжини, де можуть бути визначені як

перелік дій у кожному стані, так і умови переходів в інші стани. Для обробки цих типів може бути застосований синтаксичний аналіз. Атрибут Name з першого відношення призначений для помітки в разі переходу у стан, який ще не був описаний. Якщо перехід визначений в уже існуючі стани, то ідентифікатор переходу записується у друге відношення в атрибут Next Id_state. В разі переходу в неописаний стан в атрибут Next Id_state записується ім'я, яке у подальшому має визначитись у першому відношенні атрибутом Name. Переходи в описані (а) та неописані (б) стани графічно показані на рис. 2.

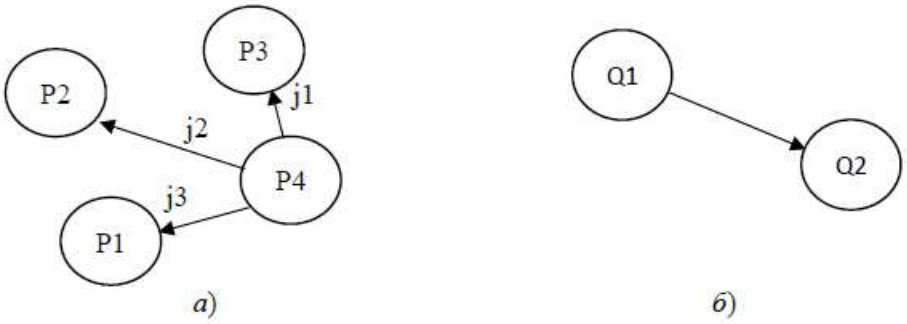

Рисунок 2 – Переходи в описані та неописані стани

У результаті опису моделі програми формується її автоматна модель, яка зберігається у вигляді бази даних. При можливості виконання паралельних гілок алгоритму опис виглядатиме таким чином: {умова і}[перелік дій] | N {j}[дії гілки 1] | N {j}[дії гілки 2] | N {j}[дії гілки 3]. Графічно частина моделі з паралельними гілками зображена на рис. 3. Кореневий стан і1 у даному прикладі породжує три стани, які можуть виконуватись паралельно. Вершина Q2 є вершиною-протоколом, яка запускається на виконання в разі завершення виконання гілок j1, j2, j3. Як бачимо, при описі паралельних гілок маємо однакові умови предикатів для станів j1, j2, j3. Саме те, що таке розгалуження має однакові умови розгалуження, вказує на те, шо це паралельні гілки, а не звичайне розгалуження, як показано на рис. 2 а).

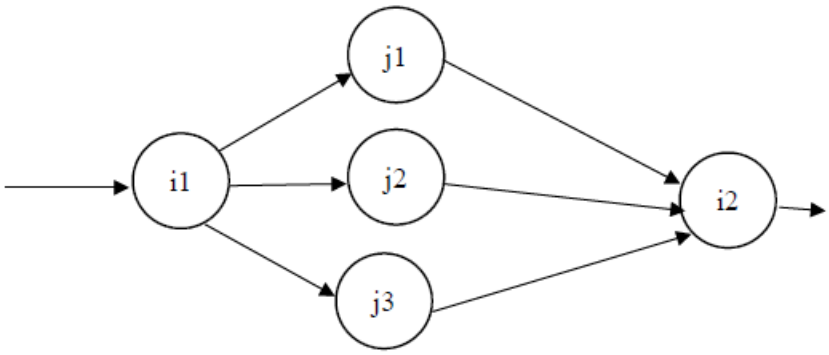

Рисунок 3 – Фрагмент моделі з паралельними гілками

У паралельних програмах часто застосовуються спільні ресурси, до яких мають доступ різні потоки та процеси. Для синхронізації доступу до спільних ресурсів використовуються монітори, які запропонував Хоар [3]. Монітор описується спеціальним станом, який забезпечує тільки один процес або потік. {p U Nq}[дії доступу до ресурсу]. Якщо монітор вільний, то забезпечується доступ до спільного ресурсу, інакше оператор Nq очікує

звільнення у стані-монітора. Графічно це ілюструє рис. 4. На ньому умовно показано вхід кількох процесів у стан-монітор. Якщо монітор зайнятий, то він знаходиться у стані очікування, інакше надає доступ до ресурсу, що показано виходом зі стану. Для опису паралельних алгоритмів останніх двох станів цілком достатньо для створення відповідної моделі. Визначення закінчення стану відбувається тоді, коли маємо конструкцію опису виду p U |q1|q2..|q_k. Тобто, на певний момент маємо розгалуження по кількох напрямах. При цьому за предикатом q1 формується один новий стан, за предикатом q2 формується інший новий стан і так само щодо предиката q_k. Таким чином, у результаті опису моделі за допомогою предикатів формується власно сама модель, яка подається у вигляді спеціальної бази даних як внутрішнє уявлення.

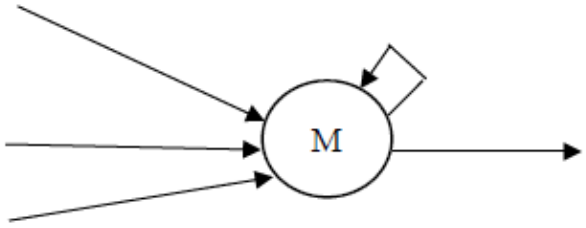

Рисунок 4 – Графічне зображення стану-монітора

Після створення автоматної моделі програми виконується крок її реалізації на обраній процедурній мові програмування. Обробка моделі починається з початкових станів. Надалі виконується просування по зв'язках до інших станів. На черговому рівні обробляється перший зв'язок і так послідовно по рівнях. По досягненні кінцевого стану або при поверненні у стан, до якого вже було звернення, відбувається повернення на попередній рівень і обробляється чергове розгалуження на цьому рівні. Атрибут Mark відношення MAIN використовується для того, щоб помітити стан, до якого вже відбувалось звернення. Перехід до обробки наступних станів доречно виконувати як звернення до підпрограми. По закінченні обробки чергового маршруту графа моделі зручно повертатись на попередній рівень як вихід із підпрограми. Процес обробки, який пов'язаний з поверненнями, наведений на рис. 5.

На рисунку пунктирними лініями показані повернення у попередні стани для того, щоб обробити всі розгалуження попередніх станів. Так, при обробці кінцевого стану F із стану A4 пунктиром показані зворотне повернення і перехід для обробки спочатку гілки у стан A41 і далі у стан A42. По завершенні обробки гілок із стану A4 пунктиром показано повернення у стан A3 і обробку гілки розгалуження і далі повернення у стан A2. Таким чином, процес обробки автоматної моделі по досягненні кінцевого стану формально можна представити як Q={розгалуження≠<пусто>}[обробка(Σa_i)] U_p{початковий стан}[stop] | N{true}[Na_{i-1}].

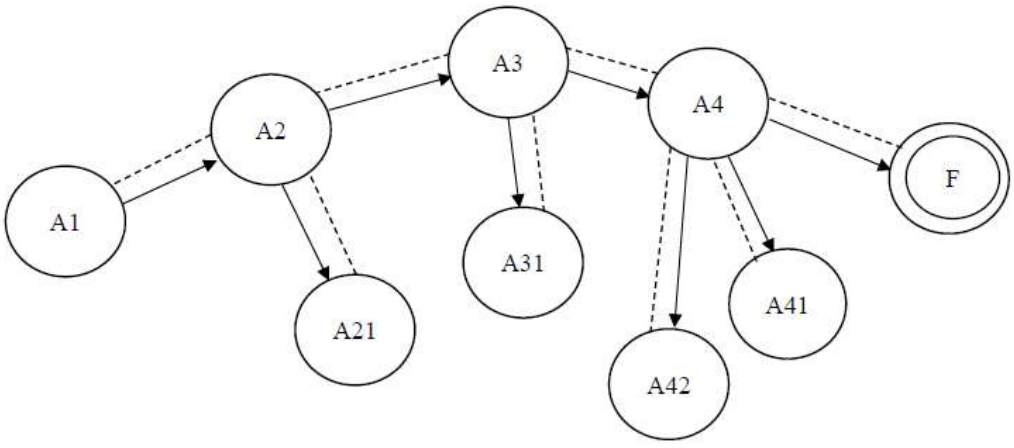

Рисунок 5 – Порядок обробки графа автоматної моделі

Отже, якщо не всі гілки розгалуження для певного стану оброблені, то виконується обробка кожної гілки. При обробці всіх розгалужень для кожного стану здійснюються повернення у попередній стан з його розгалуженнями та обробка цих розгалужень. Процес продовжується доти, доки повернення у попередній стан буде неможливим, і процес обробки моделі на цьому завершується. В результаті такої обробки графа моделі всі його гілки будуть оброблені. Власне, обробка кожного стану полягає у реалізації програми на деякій процедурній мові програмування тих дій, що мають виконуватись у даному стані.

3. Висновки

Запропонована технологія створення моделей програм передбачає побудову моделі безпосередньо при її описі. Тому, в разі коректного опису моделі, відпадає потреба у її наступній верифікації. Дії, визначені у кожному стані, перекладаються на будь-яку процедурну мову програмування. Треба додати, що процес обробки предикатів піддається автоматизації. Умови у предикатах мають розпізнаватися синтаксичним аналізатором і формувати умови переходів в інші стани при подальшому перетворенні моделі у програму. Те ж саме стосується обробки станів моделі, в яких дії в них можуть бути реалізовані операторами присвоєння, введення/виведення, функцій та ін. Крім того, в середині стану можливі локальні розгалуження і цикли. Нарешті, в залежності від вимог програмна реалізація може здійснюватися на будь-якій процедурній мові програмування. При цьому програма трансляції опису моделі програми спочатку має формувати своє внутрішнє уявлення для подальшого налаштування на конкретну мову програмування. Таким чином, у разі коректного опису моделі програми власне модель програми також буде коректною і перетворення її у програму на конкретну мову програмування може бути виконано за відомими схемами трансляції.

СПИСОК ДЖЕРЕЛ

1. Ошибка в ПО Airbus A350 вынуждает перезагружать системы самолетов каждые 149 часов. URL: https://internetua.com/oshibka-v-po-airbus-a350-vynujdaet-perezagrujat-sistemy-samoletov-kajdye-149-csasov.
2. Salapatov V. Technology for modelling software systems based non-deterministic finite automatons. Security $ future internetional scientific journal: abstracts of reports. Sofia, Bulgaria, 2019. P. 113–114.
3. Карпов Ю.Г. MODEL CHECKING. Верификация параллельных и распределенных программных систем. СПб.; БХВ-Петербург, 2010. 560 с.
4. Хоар Ч. Взаимодействующие последовательные процессы / пер. с англ. М.: Мир, 1989. 264 с.

Стаття надійшла до редакції 10.07.2021