У статті розглядаються моделі алгоритмічних конструкцій на основі модифікації мереж Петрі. Закладені основи моделювання паралельних алгоритмів для аналізу та виявлення критичних властивостей при створенні систем автоматизації тестування паралельних програм

Ключові слова: моделі паралельних алгоритмів, модифікації мереж Петрі

В статье рассматриваются модели алгоритмических конструкций на основе модификации сетей Петри. Заложены основы моделирования параллельных алгоритмов для анализа и выявления критических свойств при создании систем автоматизированного тестирования параллельных программ

Ключевые слова: модели параллельных алгоритмов, модификации сетей Петри

This article discusses a model of algorithmic designs based on modification of Petri nets. The foundations of modeling parallel algorithms for analyzing and identifying critical properties when creating a computer-aided testing of parallel programs

Keywords: models of parallel algorithms, modification of Petri nets

УДК 004.942, 004.451.45, 519.687.1

ЗАСОБИ АВТОМАТИЗОВАНОГО АНАЛІЗУ ПАРАЛЕЛЬНИХ АЛГОРИТМІВ НА ОСНОВІ МОДИФІКАЦІЙ МЕРЕЖ ПЕТРІ

О.О. Супруненко
Кандидат технічних наук, доцент*
Контактний тел.: 066-187-99-50
E-mail: ra-oks@mail.ru

А.М. Парнюк*
*Кафедра математичного і програмного забезпечення автоматизованих систем
Черкаський національний університет імені Богдана Хмельницького
бульв. Шевченка, 79, корп. №3, а.275, м. Черкаси, Україна, 18006
Контактний тел.: 066-187-99-50
E-mail: oxo_@mail.ru

1. Вступ

У зв'язку з активним розвитком багатоядерних і багатопроцесорних обчислювальних систем особливо актуальним на даний час є завдання створення паралельного програмного забезпечення, що обумовлює виникнення нових технологій створення паралельних програм. Важливим питанням при цьому є неперервне підвищення продуктивності обчислювальних систем та ефективності використання апаратних ресурсів [1]. При створенні працездатної паралельної програми програміст стикається з рядом проблем, які полягають у перетворенні звичного лінійного алгоритму програми на паралельний, врахуванні типу пам'яті та особливостей звертання до неї, в організації взаємодії паралельних процесів, у тестуванні програмного продукту на працездатність, наявність надлишковості, неоднозначностей і тупикових ситуацій, у адаптації алгоритму до динаміки завантаження вільних обчислювальних ресурсів. У зв'язку з цим важливим завданням є створення автоматизованих програмних засобів аналізу паралельних програм. Ця задача найбільш вдало вирішується із застосуванням модельного підходу [2] при використанні засобів візуального представлення топології та функціонування паралельного алгоритму у сполученні з аналітичними засобами тестування досліджуваної алгоритмічної моделі.

2. Виділення проблеми та постановка задачі

Проблеми тестування коду паралельних програм пов'язані з численними особливостями його побудови та виконання: топологією алгоритму програми, поділом на підзадачі, взаємодією підзадач, управлінням потоками на основі умов, детермінованих та недетермінованих векторів та ін. Для доказу працездатності паралельної програми недостатньо набору тестів, що покривають область значень вхідних даних. Необхідно також довести, що поведінка програми є стійкою, тобто результат роботи паралельної програми не залежить від умов виконання. Таким чином, хоча розв'язання проблеми налагодження може спростити процес розробки паралельних програм, доведення працездатності вимагає залучення математично точних методів перевірки коректності функціонування програми.

У даній роботі досліджуються моделі елементів обчислювальних алгоритмів, які побудовані з використанням аналітично-візуального засобу моделювання паралельних програм – мереж Петрі. Вибір даного за-

собу моделювання пояснюється можливістю візуального представлення паралельного алгоритму в моделі, однозначним математичним описом його функціонування та наявністю статичних і динамічних властивостей для тестування моделей паралельних програм.

3. Основне дослідження

У загальному випадку питання коректності програм перевіряється за двома групами критеріїв: 1) працездатність програми; 2) коректність вибору і реалізації алгоритму. Автоматизувати процес доказу коректності вибору алгоритму практично неможливо, оскільки для цього потрібно формалізувати задачу, розв'язувану алгоритмом у термінах, які розуміються автоматичним аналізатором, що в загальному випадку приводить до необхідності написання подібного ж алгоритму, коректність якого також необхідно доводити. Таким чином, питання коректності програми необхідно звузити до працездатності програми й коректності реалізації алгоритму. При перевірці коректності реалізації алгоритму особливу увагу варто звертати на помилки, які виникають періодично і які важко повторити по причині не детермінованості функціонування паралельних програм. Такі помилки можуть переходити у розряд прихованих, тобто таких, які погано виявляються на етапах тестування чорного ящика та відладки остаточного програмного продукту [1].

Для доведення працездатності програм протягом кількох останніх десятиліть розробляються методи й засоби верифікації програм. Незважаючи на вже існуючі засоби верифікації програм середньої складності, автоматична верифікація в наш час не є обов'язковою частиною процесу розробки програм. Але для паралельних програм, налагодження яких ускладнене внаслідок багатоваріантності виконання, верифікація програм стає особливо важливою.

Коректність реалізації алгоритму в послідовному випадку звичайно розглядають як задачу Model Checking, в якій специфікація завдання описується формулами темпоральної логіки, а програма представляється у вигляді деякої системи переходів, або як задачу Theorem Proving, в якій за допомогою автоматизованих процедур формального висновку доводиться, що результат обчислень програми відповідає поставленому завданню. У паралельному програмуванні, крім наведених властивостей коректності програми, існує необхідність доведення коректності функціонування програми, зокрема, відсутність у паралельній програмі дедлоків, досяжність кінцевого стану, а при наданні паралельною програмою інтерактивного сервісу іншим програмам – також відсутність непродуктивних циклів та живість паралельної програми.

При перевірці коректності роботи паралельних програм крім тестування й відлагодження використовують різні техніки верифікації, серед яких однією з основних є техніка модельного підходу [2, 3]. Дана техніка верифікації ґрунтується на аналізі властивостей моделей програм, виражених у термінах різних математичних теорій. Такі теорії надають точні способи й методи для верифікації, що дозволяють формально проаналізувати ті властивості паралельних програм, які цікавлять дослідника, наприклад, виявляти помилки, що виникають при організації взаємодії процесів. У рамках даної техніки верифікації для створення моделей використовують граф-схеми алгоритмів, мережені моделі, потокові діаграми й машини Тюрінга, графові моделі і мереж Петрі [1, 4]. При цьому важливу роль відіграє візуальне подання та засоби автоматизованої перевірки даних моделей.

Для вирішення поставлених проблем потрібно дібрати апарат, який дозволив би аналізувати статичні і динамічні властивості паралельної програми, сполучував наочність і можливість відображення асинхронності виконання паралельних гілок алгоритму з однозначним математичним описом функціонування. На даний момент цим критеріям цілком відповідає апарат мереж Петрі [5].

Мережі Петрі є направленим дводольним несумісним графом, що має однозначний математичний опис, дозволяє відслідковувати статичні й динамічні властивості відтворених моделей. Мережі Петрі мають численні інтерпретації та модифікації, які ділять на три основні групи: 1) числові (EN) та макрочислові мережі Петрі (MEN), 2) оціночні мережі Петрі (BPN), 3) безпечні мережі Петрі (SPN) [6].

В даній роботі розглядається побудова моделей паралельних програм на базі безпечних мереж Петрі (SPN), що обумовлено їх властивостями, які дозволяють провести прямі асоціації між елементами граф-схеми алгоритму і безпечної мережі Петрі, а також виконати автоматизований аналіз моделі. Статичні властивості SPN [6] встановлюють правила побудови моделей та оптимізації їх топології. Дані властивості SPN-мереж дозволяють відслідкувати наступні особливості при формуванні моделі:

1) дуги мережі Петрі можуть з'єднувати лише різнойменні вершини, тобто вершину переходу з вершиною місця, чи навпаки, що дозволяє контролювати логіку побудови моделі – чергування елементів, які відповідають поняттям «умова» та «функція»;

2) ділянку мережі Петрі, яка відображає низку послідовних дій, що активізуються однією міткою (автоматні мережі Петрі [6]) можна замінити одним макропереходом. Тобто, якщо в мережі Петрі, яка моделює набір послідовні дій – послідовний алгоритм – відсутні критичні властивості (тестування даної ділянки ведеться за тестами для послідовних алгоритмів), то на етапі дослідження властивостей моделі деталізувати такі ділянки не потрібно;

3) мережа Петрі вважається замкненою, якщо через кінцеве число спрацьовування переходів вона відновлює початкову розмітку.

Важливим механізмом відслідковування неоднозначностей в розгалуженнях, дедлоків, неживих гілок є три динамічні властивості мереж Петрі: безпечність, неконфліктність та живість [5].

Розглянемо окремий процес, представлений мережею Петрі. При комбінації таких процесів одержимо систему паралельних процесів. Прикладом окремого процесу може бути код деякої послідовної програми. Програма представляє два різних аспекти процесу: обчислення й керування. Обчислення пов'язане з поточними арифметичними й логічними операціями, введенням і виведенням, звичайними маніпуляціями над ділянками пам'яті і їхнім вмістом. Керування ж визначає порядок виконання обчислень.

Мережі Петрі вдало відображають керування програмними моделями. Вони призначені для моделювання впорядкування інструкцій і потоку інформації, але не для дійсного обчислення самих значень. В моделі дані потрібно зв'язати з певними елементами мереж. Тому перед створенням моделі паралельної програми, спершу необхідно сформувати примітиви, що відображають елементарні алгоритмічні конструкції: елементи вибору, цикли, виклик процедур та функцій і т.д.

Розглянемо подання цих алгоритмічних конструкцій за допомогою мереж Петрі. При трансформації PN-моделі у паралельну програми будемо розглядати алгоритмічні конструкції мови Ада. Такий вибір пояснюється наявністю вбудованих в саму мову конструкцій, що забезпечують функціонування і взаємодію паралельних процесів, що спрощує аналіз подання алгоритмічних конструкцій.

Крім того, мова Ада – єдина індустріальна мова, що надає високорівневі засоби роботи з асинхронними процесами: програмні модулі, що описують асинхронні процеси; структури даних, доступ до яких можливий тільки в режимі взаємного виключення; конструкції, що забезпечують синхронізацію процесів і обмін даними; конструкції, що дозволяють процесу вибирати варіанти поточної реалізації в залежності від ситуації та ін.

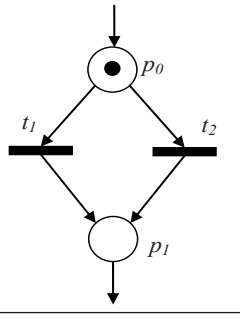
Конструкція, що наведена в табл. 1 відображає класичне подання структури умовного оператору характерне для всіх мов програмування.

Мітка, опинившись в вершині p_0, призводить до спрацювання лише одного з переходів t_1 або t_2, що відповідає класичній ситуації при виникненні розгалуження в програмі. Такою ж конструкцією відображається і спрощений варіант даного оператору, коли в коді явно не вказується, послідовність дій, що виконується після оператору else.

В даному випадку перехід буде являти собою не інструкцію чи послідовність інструкцій, а перехід на наступну частину коду, що розташована за розгалуженням.

Таблиця 1

```
if <condition> then
 <statements>
else
 <statements>
end if;
```

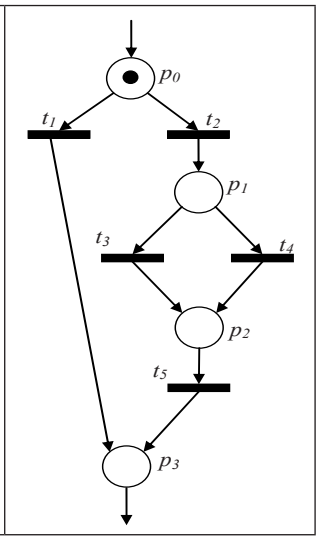


При відображенні конструкції умовного оператору мережею Петрі потрібно потурбуватися про управління вершиною переходу p_0 для уникнення однієї з критичних ситуацій, неоднозначності у виконанні інструкцій – конфліктності.

Для цього застосуємо функціональну модифікацію мереж Петрі, яка розроблена на базі безпечних управляючих мереж Петрі. Вона дозволяє керувати вершинами переходів і реалізовувати вибір одного варіанту з багатьох у розгалуженні [5], що важливо також для конструкції каскадного розгалуження, в якому окрім else присутній ще оператор elseif з умовою (табл. 2).

Таблиця 2

```
if <condition> then
 <statements>
elsif <condition> then
 <statements>
else
 <statements>
end if;
```

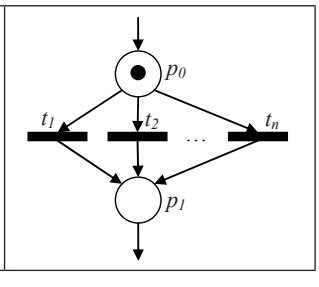


Даний оператор може повторюватись коді кілька разів. В такому випадку, наприклад, додавання ще одного elseif, приведе до відповідної зміни шаблону в табл. 2 шляхом додаткового розгалуження в одній із гілок (після переходу t_4), куди буде додано вже розглянуту вище конструкцію простого розгалуження.

До конструкцій розгалуження можна також віднести оператор вибору, де кожна з альтернатив виконання представляється переходом:

Таблиця 3

```
case <expression> is
   when <choice>=>
 <statements>
   when <choice>=>
 <statements>
   ....
   when others =>
 <statements>
end case;
```

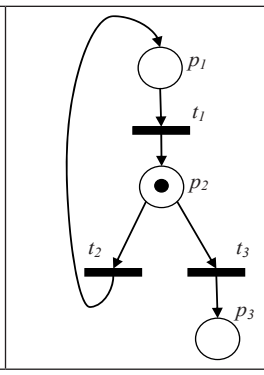


Деяку складність при моделюванні представляє інструкція переходу goto.

Вона передбачена для використання в мові Ада у виняткових ситуаціях, і в загальному випадку має два варіанти застосування: перехід на попередню частину програмного коду (табл. 4.1) і на код, що розташований нижче (табл. 4.2). Оскільки використання інструкції goto дуже обмежене (не можна здійснити перехід всередину умовної інструкції if, усередину циклу (loop), або, як у мові Паскаль, за межі підпрограми), то в першому випадку, дана конструкція, будучи представлена мережею Петрі, візуально і логічно аналогічна циклу (табл. 4.1), в другому – оператору розгалуження (табл. 4.2).

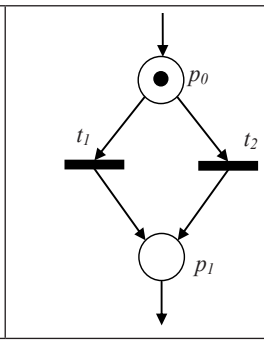
Таблиця 4.1

```
begin
<<label>>
 <statement>
goto <<label>>;
end;
```



Таблиця 4.2

```
begin
goto <<label>>;
 <statement>
<<label>>
end;
```



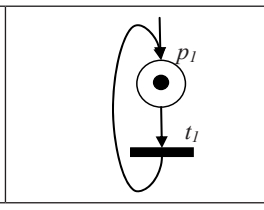
Очевидно, що на практиці застосування goto потребує додаткового використання операторів умовного переходу (if) для перевірки деякої умови, а також наявності мітки. Все це призводить до ускладнень при побудові реальної моделі, оскільки не можна явно визначити конкретні елементи мережі Петрі, що відповідають даному оператору.

Розглянемо наступний тип конструкцій без яких практично не обходиться жодна програма – цикли. Мова Ада надає три стандартні структури для їх виконання.

Прикладом найпростішого циклу на мові Ада може послужити безкінечний цикл.

Таблиця 5

```
loop
 <statements>
end loop;
```



Мітка, що знаходиться у вершині місця позначає готовність до виконання певних дій, перехід – деяку послідовність операторів, що виконується всередині циклу. Після спрацювання переходу, мітка знову повертається в початкову вершину і так до безкінечності.

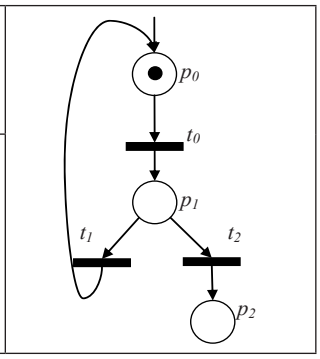
Проте застосування циклічних конструкцій даного типу найчастіше комбінується разом з інструкцією виходу exit, що розглядається нижче і забезпечує завершення циклу.

Наведені в табл. 6 варіанти коду в термінах мережі Петрі мають однакову модель. Інструкції exit в операторі розгалуження і exit when можуть бути використані для передчасного виходу із циклу. При цьому, виконання програми буде продовжено в точці, що безпосередньо розташована за циклом.

Таблиця 6

```
loop
 <statements>
exit when <condition>;
 <statements>
end loop;
```

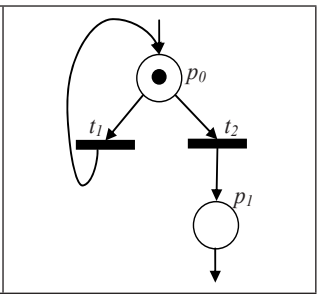
```
loop
 <statements>
 if <condition> then
 exit;
 end if;
 <statements>
end loop;
```



У простому циклі з передумовою while, що ідентичний циклу while мови Паскаль, перевірка умови виконання циклу відбувається до входу в блок інструкцій з якого складається тіло циклу (табл. 7):

Таблиця 7

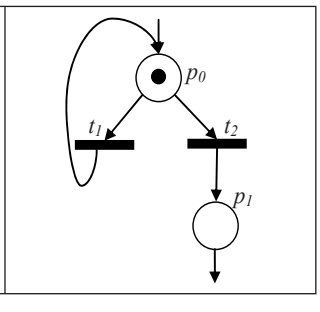
```
while <condition> loop
 <statements>
end loop;
```



Конструкція циклу з параметром може бути побудована за допомогою мережі Петрі аналогічно до конструкції повторення з передумовою (табл. 8). Дійсно, даний цикл є особливою конструкцією циклу з передумовою, у якій зміна і перевірка параметру відбувається автоматично на рівні заголовку циклу.

Таблиця 8

```
for <scheme> loop
 <statements>
end loop;
```



В програмах часто виникають ситуації коли дані алгоритмічні конструкції є вкладеними одна а одну. Для таких випадків будується каскадна модель з конструкцій розглянутих вище.

Розглянемо правила об'єднання шаблонів для побудови моделей у термінах мереж Петрі в цілісне подання. Їх можна сформулювати так:

• Кожна мережа (як і кожний шаблон) починається і закінчується вершиною місця. Дане правило зумовлене логікою мереж Петрі в яких вершина місця відповідає певному стану мережі (готовністю до виконання певних дій або свідченням завершення дії), а перехід – безпосередньому виконанню дії.

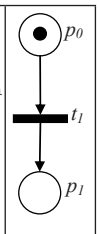
• Здійснення певних обчислень в програмі (арифметичних і логічних операцій, введення і виведення, маніпуляції над ділянками пам'яті і їхнім вмістом і т.д.) та інших дій, що не передбачають використання управляючих структур відображається простим переходом. Це правило стосується також виклику стандартних процедур і функцій.

З огляду на це найпростіший приклад мережі, що відображає деяку програми, матиме вигляд, поданий у табл. 9.

Таблиця 9

```
with Ada.Text_IO; use Ada.Text_IO;

procedure Hello(...)is
I: Integer := 1; -- описова частина
  . . .
begin
  . . .
  I := I * 2; -- послідовність
інструкцій
  Put_Line("Hello World!"); --
виклик процедури
  . . .
end Hello;
```



• При вкладеності одного шаблону в інший перехід всередині відповідної конструкції замінюється наступними елементами: перехід → вершина місця (що позначає початок вкладеного шаблону) → вкладений шаблон → вершина місця (що означає закінчення вкладеного шаблону) → перехід, після яких відображається структура тієї частини шаблону, перед якою розмістилася вкладена конструкція.

Наприклад, на рис. 1 вкладена структура позначена для наочності макропереходом з явним виділенням початкової і кінцевої вершин.

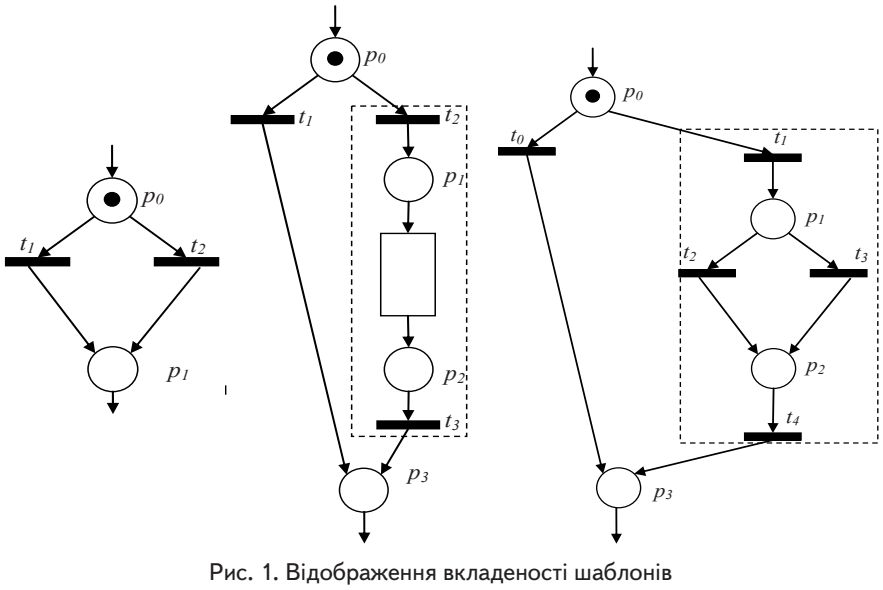


Рис. 1. Відображення вкладеності шаблонів

Робота з паралельними процесами в мові Ада здійснюється за допомогою задач, виконання яких відбувається паралельно. Задачі є програмними модулями мови Ада. Задачі, як підпрограми й пакети, мають дві частини: специфікацію й тіло. Як у випадку підпрограм, так і пакетів, специфікація задачі з наступним символом «;» становить оголошення задачі. На противагу підпрограмам і пакетам задачі не можуть бути скомпільовані самостійно. Тому для компіляції задачі мають бути розміщені у підпрограму або пакет [7].

Нижче наведено опис специфікації і тіла задачі:

```
task [type] ідентифікатор_задачі is
    --описи входів
end ідентифікатор_задачі;

task body ідентифікатор_задачі is
    -- декларативна частина
begin
    --тіло оператору прийняття входів
    --послідовність_операторів
end ідентифікатор_задачі;
```

Засобами мереж Петрі легко моделюється створення й виконання паралельних задач. Наприклад, на рис. 2 зображені досить популярні на сьогоднішній день конструкцій fork/join. Перехід fork моделює «розгалуження» – створення з однієї гілки (потоку), що виконується, двох паралельних гілок. Це, як правило, реалізується шляхом додавання однієї додаткової гілки до існуючої. Перехід join, у свою чергу, здійснює «злиття» двох гілок (потоків) по завершенню їхньої роботи – знищення створеної паралельної гілки. [8].

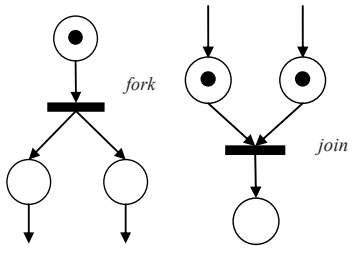


Рис. 2. Конструкції розгалуження і злиття задач

Якщо в тілі програми присутні описи специфікацій і тіла задач, то при активації даних задач у відповідному місці, в залежності від того, в якій структурі описано задачі, відбудеться розпаралелювання на відповідну кількість гілок мереж Петрі (операція fork). Тобто для кожної нової задачі створюватиметься нова гілка. Якщо специфікації задач не містять описів входів, переходи кожної гілки будуть спрацьовувати незалежно від інших гілок, до місця логічного звершення задач (операція join).

Таблиця 10

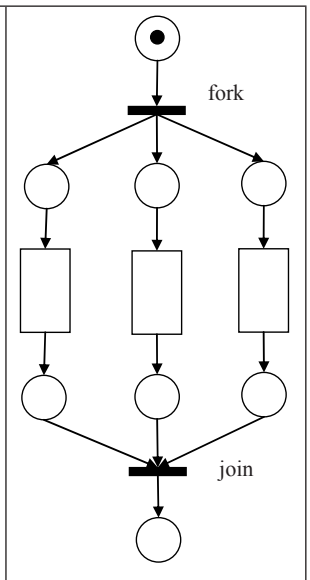
```
with Ada.Text_IO;
procedure Multitasking_Demo is

 task Anonimous_Task;
 task body Anonimous_Task is
 begin
 for Count in 1..5 loop
 Ada.Text_IO.Put_Line("Hello");
 end loop;
 end Anonimous_Task;

 task type Simple_Task (M: Character);
 task body Simple_Task is
 begin
 for Count in 1..5 loop
 Ada.Text_IO.Put_Line("SlTask" & M);
 end loop;
 end Simple_Task;

Simple_Task_Var:Simple_Task(M=>'A');

begin
 null;
end Multitasking_Demo;
```



Розглянемо простий приклад створення, активації і завершення роботи двох паралельних задач. Для спрощення в прикладі послідовність певних операцій в тілі задачі позначається прямокутником (табл. 10). У випадку динамічно створених задач розгалуження (створення нової гілки) відбувається в місці використання оператору new.

Таким чином моделюється додавання до основної програми ще двох паралельно виконуваних задач. Перехід join, у свою чергу, здійснює «злиття» трьох гілок по закінченню їхньої роботи, що відбудеться коли виконається тіло задачі.

4. Висновки

Дана робота присвячена розробці примітивів у термінах мереж Петрі, які відповідають характерним алгоритмічним конструкціям обраної програмної технології і дозволяють будувати моделі паралельних програм, відображати їх роботу та проводити тестування. Використання мережевої топології моделі дозволяє вдало сполучувати фізичне відображення, логічний опис та керування обміном даними/командами [9]. В даній роботі закладаються основи для дослідження задачі автоматизованого пошуку проблемних ділянок в алгоритмах паралельних програм. Властивості мереж Петрі дозволяють проводити перевірку побудованих моделей на неконфліктність, безпечність та живість, що дозволяє частково автоматизувати непростий етап розробки паралельних програм – тестування.

В якості досліджуваної програмної технології було обрано мову програмування Ада, що характеризується вбудованою підтримкою багатозадачності, яка забезпечується не за допомогою певних розширень або зовнішніх бібліотек, а за допомогою строго стандартизованих засобів, які вбудовані безпосередньо в мову програмування. Розглянуті алгоритмічні конструкції характерні для більшості мов програмування, тому PN-моделі можуть застосовуватися для їх моделювання та аналізу.

Література

1. Карпов А. Тестирование параллельных программ. [Электронный документ]. Режим доступа: http://www.viva64.com/content/articles/parallel-programming/?f=Parallel_program_testing_rus.html&lang=ru&content=parallel-programming. Проверено: 14.07.2010.
2. Бородакий В.Ю., Окороченко Г.Е. Анализ средств имитационного моделирования распределённых информационных систем. // Компьютерные системы и технологии: Научная сессия МИФИ. – 2007. – Том 12. – С. 129-130.
3. Евгеньев Б.Г. Модели вместо алгоритмов. Смена парадигмы разработки прикладных систем. // Наука и образование. – 2005. – №9. – [Электронный документ]. Режим доступа: http://technomag.edu.ru/doc/56632.html. Проверено: 16.07.2010.
4. Голенков Е.А., Соколов А.С., Метод автоматического построения модели параллельной программы в терминах сетей Петри. // Вычислительные методы и программирование. – 2005. – Т6. – №2. – С. 77-82.
5. Кузьмук В.В., Супруненко О.О. Расширение функциональных возможностей алгоритмического аппарата сетей Петри при моделировании параллельных процессов. // Электронное моделирование. –2009.–Т.31.–№5.– С.65-73.
6. Кузьмук В.В. Методика алгоритмического описания и моделирования параллельных процессов управления. – К.: Наукова думка, 1981. – 56 с.
7. Вегнер П. Программирование на языке Ада: Пер. под ред. В.Ш.Кауфмана. – М.: Мир, 1983 - 256 с.
8. Федотов И.Е. Некоторые приемы параллельного программирования: Учебное пособие. – М.: Наука, 2008. – 188 с.
9. Супруненко О.О., Братко О.В. Case-засоби автоматизованого аналізу програм на основі мереж Петрі. // Вісник Черкаського державного технологічного університету. – 2009. – № 3. – С. 12-15.