MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 28, NO. 4, 2021
journal homepage: www.mais-journal.ru

THEORY OF COMPUTING

A Mathematical Model of Parallel Programs and an Approach Based on it to Verification of MPI Programs

A. M. Mironov[1]

DOI: 10.18255/1818-1015-2021-4-394-412

[1]Lomonosov Moscow State University, 1 Leninskie Gory, Moscow 119991, Russia.

MSC2020: 68Q60
Research article
Full text in Russian

Received November 15, 2021
After revision December 1, 2021
Accepted December 8, 2021

The paper presents a new mathematical model of parallel programs, on the basis of which it is possible, in particular, to verify parallel programs presented on a certain subset of the parallel programming interface MPI. This model is based on the concepts of a sequential and distributed process. A parallel program is modeled as a distributed process in which sequential processes communicate by asynchronously sending and receiving messages over channels. The main advantage of the described model is the ability to simulate and verify parallel programs that generate an indefinite number of sequential processes. The proposed model is illustrated by the application of verification of the matrix multiplication MPI program.

Keywords: parallel programs; MPI; distributed processes; verification

INFORMATION ABOUT THE AUTHORS

Andrew M. Mironov	orcid.org/0000-0002-9132-7804. E-mail: amironov66@gmail.com
correspondence author	Associate professor.

Funding: Grant from the Ministry of Digital Development, Communications and Mass Media of the Russian Federation and JSC "Russian Venture Company" (contract No. 004/20 from 20.03.2020, IGK 0000000007119P190002) .

For citation: A. M. Mironov, "A Mathematical Model of Parallel Programs and an Approach Based on it to Verification of MPI Programs ", *Modeling and analysis of information systems*, vol. 28, no. 4, pp. 394-412, 2021.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 28, № 4, 2021
сайт журнала: www.mais-journal.ru

THEORY OF COMPUTING

Математическая модель параллельных программ и основанный на ней подход к верификации MPI-программ

А. М. Миронов[1]

DOI: 10.18255/1818-1015-2021-4-394-412

[1]Московский государственный университет им. М. В. Ломоносова, Ленинские горы, д. 1, г. Москва, 119991 Россия.

УДК 519.681.2
Научная статья
Полный текст на русском языке

Получена 15 ноября 2021 г.
После доработки 1 декабря 2021 г.
Принята к публикации 8 декабря 2021 г.

В работе излагается новая математическая модель параллельных программ, на базе которой можно в частности верифицировать параллельные программы, представленные на некотором подмножестве программного интерфейса параллельного программирования MPI. Данная модель основана на понятиях последовательного и распределенного процесса. Параллельная программа моделируется распределенным процессом, в котором последовательные процессы взаимодействуют путем асинхронной передачи и приема сообщений через каналы. Главным преимуществом изложенной модели является возможность моделирования и верификации параллельных программ, порождающих неопределенное число последовательных процессов. Изложенная модель проиллюстрирована применением к верификации MPI программы перемножения матриц.

Ключевые слова: параллельные программы; MPI; распределенные процессы; верификация

ИНФОРМАЦИЯ ОБ АВТОРАХ

Андрей Михайлович Миронов
автор для корреспонденции

orcid.org/0000-0002-9132-7804. E-mail: amironov66@gmail.com
доцент.

Финансирование: Грант министерства цифрового развития, связи и массовых коммуникаций РФ и АО «Российская венчурная компания» (договор No004/20 от 20.03.2020, ИГК 0000000007119P190002).

Для цитирования: A. M. Mironov, “A Mathematical Model of Parallel Programs and an Approach Based on it to Verification of MPI Programs ”, *Modeling and analysis of information systems*, vol. 28, no. 4, pp. 394-412, 2021.

1. Введение

Параллельные программы – это программы, предназначенные для исполнения на многопроцессорных вычислительных системах (МПВС). Проблема разработки корректных и безопасных параллельных программ представляет в настоящее время исключительно высокую актуальность. Формальное обоснование свойств корректности и безопасности параллельных программ (называемое также **верификацией** параллельных программ) является сложной математической задачей. Существующие методы решения данной задачи пригодны лишь для достаточно ограниченного класса параллельных программ.

Одним из наиболее широко используемых средств для описания параллельных программ является программный интерфейс MPI (Message Passing Interface). В настоящей работе вводится новая математическая модель параллельных программ, на основе которой можно решать задачи верификации параллельных программ, представленных на некотором подмножестве MPI. Введенная модель иллюстрируется применением к решению задачи верификации MPI-программы умножения матриц. Наиболее близким из существующих в настоящее время подходов к моделированию параллельных программ к тому подходу, который излагается в настоящей работе, является формализм расширенных конечных автоматов (Extended Finite State Machine – EFSM) [1].

Наиболее важной особенностью предлагаемого подхода является возможность его применения для верификации параллельных программ, которые могут порождать неопределенное число процессов. Среди других подходов к моделированию и верификации таких программ следует отметить подход в [2]. В этой работе представлен инструмент ParTypes для моделирования и верификации параллельных программ, порождающих неопределенное число процессов. К сожалению, данный инструмент не работает для параллельных программ со свойством wildcard receives, которое имеет следующий смысл: в параллельной программе допускается использование действий приема сообщений от произвольного процесса (т.е. при выполнении действия такого типа номер процесса, от коротого принимается сообщение, можно определить лишь после выполнения этого действия). В частности, при помощи подхода, лежащего в основе данного инструмента, невозможно верифицировать MPI-программу для умножения матриц, рассматриваемую в настоящей статье. Среди работ, посвященных верификации MPI-программ, следует также отметить работу [3], в которой рассматривается пример верификации MPI-программы, основанный на применении аппарата машин абстрактных состояний (ASM) [4]. Существуют и другие подходы с использованием символьного исполнения и проверки модели, см. например [5—12], однако все эти подходы пригодны лишь для анализа параллельных программ, порождающих заранее заданное количество процессов.

В настоящей работе рассматривается решение задачи верификации конкретной MPI-программы, вычисляющей произведение двух матриц. Отметим, что верифицируется лишь модель данной программы, в которой умножение чисел предполагается не приближенным, а точным. Подход к верификации, представленный в работе, связан с преобразованием MPI-программы в распределенный процесс. В работе не описывается общий метод верификации, пригодный для верификации достаточно широкого класса MPI-программ, и нет описания перспектив его автоматизации. Тем не менее, преимуществом данной работы является пример решения задачи верификации такой MPI-программы, которую невозможно верифицировать на основе использования других известных моделей параллельных программ.

2. Необходимые сведения по MPI

2.1. MPI-программы

MPI (Message Passing Interface) представляет собой набор функций, типов и констант, позволяющих создавать программы (называемые **MPI-программами**) для МПВС.

Выполнение MPI-программы на МПВС заключается в том, что на каждом из узлов, входящих в эту систему, порождается вычислительный процесс, соответствующей этой MPI-программе. Все процессы, порожденные MPI-программой на узлах МПВС, функционируют параллельно, и могут обмениваться информацией друг с другом посредством **передачи сообщений**. Совокупность всех процессов, порожденных MPI-программой, обозначается записью `MPI_COMM_WORLD`.

Каждый из процессов, порожденных MPI-программой, имеет номер (называемый **рангом**), являющийся числом из множества $\{0, \dots, m-1\}$, где m – количество процессов, порожденных MPI-программой.

MPI содержит функции `MPI_Comm_rank` и `MPI_Comm_size`, позволяющие каждому из процессов, порождаемых MPI-программой, узнать свой ранг и количество запущенных процессов, порожденных этой MPI-программой, соответственно. Эти функции имеют следующий формат.

- `int rank;`
  `MPI_Comm_rank (MPI_COMM_WORLD, &rank);`
  После выполнения данной функции значение переменной `rank` будет равно рангу процесса, вызвавшего эту функцию.
- `int nprocs;`
  `MPI_Comm_size (MPI_COMM_WORLD, &nprocs);`
  После выполнения данной функции значение переменной `nprocs` будет равно количеству процессов, порожденных MPI-программой.

Процессы, порожденные какой-либо MPI-программой, могут выполняться по-разному из-за того, что результаты вызова `MPI_Comm_rank` в этих процессах будут различными.

2.2. MPI-функции передачи сообщений

Под **сообщением** в MPI понимается массив данных определенного типа, а под **передачей сообщения (ПС)** – действие, в результате которого сообщение одного процесса пересылается другому процессу. В настоящем тексте мы будем рассматривать лишь **асинхронный** режим ПС, который заключается в том, что процесс, пославший сообщение, после посылки сообщения не приостанавливает свою работу (переходя в режим ожидания доставки этого сообщения), а посланное сообщение помещается в очередь, из которой оно затем будет взято процессом-получателем.

Мы будем рассматривать MPI-функции ПС следующих двух видов.

- Функции **попарной ПС (ППС)**.
  В ППС участвуют только два процесса, один из них является отправителем сообщения, а другой – получателем этого сообщения.
- Функция **коллективной ПС (КПС)**.
  В рассматриваемой функции КПС участвуют все процессы, порожденные MPI-программой, процесс с рангом 0 является отправителем сообщения, а остальные процессы являются получателями.
  Сообщения, посылаемые функциями КПС, не могут быть получены функциями ППС, и наоборот.

Ниже мы приводим описания некоторых MPI-функций ПС. В объяснениях смысла этих функций после имени каждого аргумента мы указываем в скобках его тип.

1. Посылка сообщения (ППС):

$$\begin{array}{ll} \texttt{MPI\_Send} & (p, n, \tau, \\ & r, l, \texttt{MPI\_COMM\_WORLD}); \end{array} \tag{1}$$

   Выполнение данной функции заключается в посылке сообщения процессу с рангом r (`int`). Посылаемое сообщение представляет собой массив из n (`int`) элементов типа τ (`MPI_Datatype`),

начало которого находится по адресу p (`void *`). К посылаемому сообщению присоединяется тег (т.е. метка) l (`int`).

2. Получение сообщения (ППС):

$$\begin{aligned} \texttt{MPI\_Recv} \quad & (p, n, \tau, \\ & \texttt{MPI\_ANY\_SOURCE}, \texttt{MPI\_ANY\_TAG}, \\ & \texttt{MPI\_COMM\_WORLD}, q); \end{aligned} \tag{2}$$

Выполнение данной функции заключается в получении сообщения, которое должно быть размещено в участке памяти по адресу p. Предполагается, что получаемое сообщение представляет собой массив из не более n элементов типа τ.
q (`MPI_Status *`) – адрес структуры, в которой должна быть размещена информация о принятом сообщении. Эта структура содержит поля: `MPI_SOURCE` (в нем должен быть размещен ранг отправителя), `MPI_TAG` (в нем должен быть размещен тег принятого сообщения), и другие поля.

3. Рассылка сообщения из процесса с рангом 0 остальным процессам (КПС):

$$\begin{aligned} \texttt{MPI\_Bcast} \quad & (p, n, \tau, \\ & 0, \texttt{MPI\_COMM\_WORLD}); \end{aligned} \tag{3}$$

Эта функция выполняется следующим образом.

- В процессе с рангом 0 происходит посылка всем остальным процессам сообщения, которое представляет собой массив из n (`int`) элементов типа τ (`MPI_Datatype`), начало которого находится по адресу p (`void *`).
- В остальных процессах происходит получение от процесса с рангом 0 сообщения, которое представляет собой массив из n элементов типа τ, и должно быть размещено в участке памяти по адресу p.

3. MPI-программа умножения матриц

В этом пункте мы излагаем пример MPI-программы умножения матриц. Задача умножения матриц заключается в том, чтобы по заданным матрицам A и B вычислить их произведение $C = AB$. Данный пример используется ниже для иллюстрации применения излагаемой в настоящей работе модели параллельных программ для решения задачи верификации MPI-программ.

3.1. Неформальное описание MPI-программы умножения матриц

Неформально работу излагаемой в этом пункте MPI-программы для умножения матриц A и B можно описать следующим образом. Мы будем называть процесс с рангом 0 **менеджером**, а остальные процессы – **работниками**. Работа менеджера состоит из следующих действий:

- рассылка второй матрицы (B) всем работникам,
- назначение задач работникам, и
- прием результатов от работников.

Каждая задача работнику заключается в вычислении одной строки матрицы $C = AB$. Менеджер назначает эту задачу путем посылки работнику сообщения, содержащего одну строку матрицы A. Тег этого сообщения равен номеру посылаемой строки.

Как только менеджер получает от работника результат (т.е. сообщение с вычисленной строкой произведения, его тег равен номеру этой строки), он посылает этому работнику

- новую задачу (если еще есть неназначенные задачи), или
- сообщение с тегом 0 (если неназначенных задач нет).

3.2. MPI-программа умножения матриц

Излагаемая ниже MPI-программа для умножения матриц использует вспомогательную функцию `vecmat` умножения строки `vector[L]` на матрицу `matrix[L][M]` и записи результата в массив `result[M]`. Данная функция имеет следующий вид:

```
void vecmat (double vector[L],
             double matrix[L][M],
             double result[M])
{ int j, k;
  for (j = 0; j < M; j++)
    for (k = 0, result[j] = 0.0; k < L; k++)
      result[j] += vector[k]*matrix[k][j];
}
```

В излагаемой ниже MPI-программе будем использовать следующие обозначения: `input(a,b);` и `output(c);` являются сокращенными записями операторов чтения из файла матриц-сомножителей и записи в файл матрицы-произведения соответственно.

MPI-программа умножения матриц A и B имеет следующий вид (в этой программе слева от каждой строки мы указываем ее номер, это необходимо для описания соответствия между компонентами данной программы и компонентами модели, соответствующей этой программе):

```
01 #define comm MPI_COMM_WORLD
02
03 int main(int argc, char *argv[])
04 { int rank, nprocs, i, j;
05   MPI_Status status;
06
07   MPI_Init(&argc, &argv);
08   MPI_Comm_size(comm, &nprocs);
09   MPI_Comm_rank(comm, &rank);
10
11   if (rank == 0)
12   { int count;
13     double a[N][L], b[L][M], c[N][M], tmp[M];
14
15     input(a, b);
16     MPI_Bcast(b, L*M, MPI_DOUBLE,
17               0, comm);
18     for (count = 0;
19          count < nprocs-1 && count < N;
20          count++)
21       MPI_Send(&a[count][0], L, MPI_DOUBLE,
22                count+1, count+1, comm);
23     for (i = 0; i < N; i++)
24     { MPI_Recv(tmp, M, MPI_DOUBLE,
25                MPI_ANY_SOURCE, MPI_ANY_TAG,
26                comm, &status );
27       for (j = 0; j < M; j++)
```

```
28          c[status.MPI_TAG-1][j] = tmp[j];
29        if (count < N)
30        { MPI_Send(&a[count][0], L, MPI_DOUBLE,
31                   status.MPI_SOURCE, count+1, comm);
32          count++;
33        }
34      }
35      for (i = 1; i < nprocs; i++)
36        MPI_Send(NULL, 0, MPI_INT,
37                 i, 0, comm);
38      output (c);
39    }
40
41    else
42    { double b[L][M], in[L], out[M];
43
44      MPI_Bcast(b, L*M, MPI_DOUBLE,
45                0, comm);
46      while (1)
47      { MPI_Recv(in, L, MPI_DOUBLE,
48                 0, MPI_ANY_TAG,
49                 comm, &status);
50        if (status.MPI_TAG == 0) break;
51        vecmat(in, b, out);
52        MPI_Send(out, M, MPI_DOUBLE,
53                 0, status.MPI_TAG, comm);
54      }
55    }
56
57    MPI_Finalize();
58    return 0;
59  }
```

3.3. Вспомогательные обозначения

Для удобства моделирования и анализа данной программы мы введем специальные обозначения для некоторых используемых в ней объектов:

- массивы `a[N][L]`, `b[L][M]`, `c[N][M]` будем обозначать символами A, B, C и интерпретировать их как соответствующие матрицы,
- сообщение, пересылаемое функциями `MPI_Send` в строках 21, 22 и 30, 31 программы, будем понимать как соответствующую строку матрицы A, и обозначать ее записью A_i, где i = `count` + 1,
- массив `c[status.MPI_TAG-1][M]`, в который копируется сообщение, принятое функцией `MPI_Recv` в строках 24, 25, 26, будем понимать как соответствующую строку матрицы C, и обозначать ее записью C_l, где l =`status.MPI_TAG`,
- из определения функции `vecmat` следует, что массив `out`, вычисляемый в результате выполнения функции `vecmat` в строке 51, соответствует произведению строки Y, соответствующей массиву `in`, на матрицу B, будем обозначать в модели MPI-программы данный массив `out` записью YB.

3.4. Спецификация программы умножения матриц

Спецификация изложенной выше MPI-программы умножения матриц представляет собой следующее утверждение: после завершения выполнения этой программы должно быть верно утверждение $C = AB$, которое эквивалентно утверждению

$$\forall\, i = 1, \dots, N \quad C_i = A_i B. \tag{4}$$

4. Модель параллельных программ

В этом пункте мы излагаем модель параллельных программ. Данная модель позволяет формально представлять MPI-программы, в которых используются описанные выше операторы пересылки сообщений.

Основными понятиями данной модели являются понятия последовательного и распределенного процессов. Последовательный процесс является моделью вычислительного процесса, порождаемого параллельной программой на каком-либо узле МПВС, а распределенный процесс является моделью всей параллельной программы. Предложенная модель является теоретической основой для решения задач верификации параллельных программ.

4.1. Вспомогательные понятия

4.1.1. Типы, переменные, функциональные символы, значения, термы, формулы, связывания

Предполагаем, что заданы множества $\mathcal{T}$, $\mathcal{X}$ и $\mathcal{F}$, элементы которых называются **типами**, **переменными**, и **функциональными символами (ФС)**, соответственно. Каждому элементу x множеств $\mathcal{X}$ и $\mathcal{F}$ сопоставлен некоторый тип $\tau_x \in \mathcal{T}$. $\forall f \in \mathcal{F}$ τ_f имеет вид

$$(\tau_1, \dots, \tau_n) \longrightarrow \tau, \quad \text{где } \tau_1, \dots, \tau_n, \tau \in \mathcal{T}. \tag{5}$$

$\forall \tau \in \mathcal{T}$ задано множество $\mathcal{D}_\tau$ **значений** типа τ. Символ $\mathcal{D}$ обозначает множество значений всех типов.

$\mathcal{T}$ содержит следующие типы:

- **B** (булевский тип), $\mathcal{D}_\mathrm{B} = \{0, 1\}$,
- **N** (натуральный тип), $\mathcal{D}_\mathrm{N} = \{0, 1, \dots\}$,
- **C**, значения этого типа называются **каналами**.

Множество $\mathcal{E}$ **термов** определяется индуктивно. Каждому терму e сопоставлен тип $\tau_e \in \mathcal{T}$. Определение терма имеет следующий вид:

- $\forall \tau \in \mathcal{T}$ каждое значение типа τ является термом типа τ,
- каждая переменная $x \in \mathcal{X}$ является термом типа τ_x,
- если $f \in \mathcal{F}$, $e_1, \dots, e_n$ – термы, и $\tau_f = (\tau_{e_1}, \dots, \tau_{e_n}) \longrightarrow \tau$, то запись $f(e_1, \dots, e_n)$ является термом типа τ.

$\forall f \in \mathcal{F}$, если τ_f имеет вид (5), то этому ФС сопоставлена функция (обозначаемая тем же символом f) вида $\mathcal{D}_{\tau_1} \times \dots \times \mathcal{D}_{\tau_n} \longrightarrow \mathcal{D}_\tau$.

Если $e \in \mathcal{E}$ и $\mathcal{X}_e = \emptyset$, то с e связано **значение** $value(e) \in \mathcal{D}_{\tau_e}$, которое определяется индуктивно:

- если $e \in \mathcal{D}_{\tau_e}$, то $value(e) = e$, и
- если $e = f(e_1, \dots, e_n)$, то $value(e) = f(value(e_1), \dots, value(e_n))$.

Ниже, как правило, будем обозначать значения термов без переменных теми же записями, которыми обозначаются сами эти термы.

Будем считать, что

- $\forall n \geq 1$ $\mathcal{F}$ содержит ФС $tuple_n$, позволяющий строить кортежи: для каждого списка термов $e_1, \dots, e_n$ множество $\mathcal{E}$ содержит терм $tuple_n(e_1, \dots, e_n) \in \mathcal{E}$, который будем обозначать $(e_1, \dots, e_n)$ и интерпретировать как кортеж термов $e_1, \dots, e_n$,

- $\mathcal{F}$ содержит ФС *channel* типа $\mathbf{N} \longrightarrow \mathbf{C}$, $\forall\, i \geq 0$ канал $channel(i)$ будем называть ***i*–м каналом**, терм вида $channel(e)$ будем обозначать записью c_e,
- $\mathcal{D}_\mathbf{C}$ содержит канал $\circ$, который будем называть **широковещательным каналом**, он отличается от всех каналов вида c_i.

Будем использовать следующие обозначения:

- $\mathcal{B}$ и $\mathcal{C}$ обозначают множества $\mathcal{E}_\mathbf{B}$ и $\mathcal{E}_\mathbf{C}$ соответственно, термы из $\mathcal{B}$ называются **формулами**,
- $\forall\, e \in \mathcal{E}\ \ \mathcal{X}_e = \{x \in \mathcal{X} \mid x \text{ входит в } e\}$,
- $\forall\, X \subseteq \mathcal{X}\ \ \mathcal{E}(X) = \{e \in \mathcal{E} \mid \mathcal{X}_e \subseteq X\}$, $\mathcal{B}(X) = \mathcal{E}(X) \cap \mathcal{B}$,
- $\forall\, E \subseteq \mathcal{E}\ \ \forall\, \tau \in \mathcal{T}\ \ E_\tau = \{e \in E \mid \tau_e = \tau\}$.

Ниже для каждой рассматриваемой функции вида $f : E \longrightarrow E'$, где $E, E' \subseteq \mathcal{E}$, будем предполагать, что $\forall\, e \in E\ \ \tau_{f(e)} = \tau_e$.

Запись $\mathcal{D}^*$ обозначает совокупность всех кортежей вида $(d_1, \ldots, d_n)$, где $n \geq 0$ и $d_1, \ldots, d_n \in \mathcal{D}$, в случае $n = 0$ соответствующий кортеж называется **пустым** и обозначается символом ε. Элементы $\mathcal{D}^*$ будем называть **очередями**. $\forall\, D = (d_1, \ldots, d_n) \in \mathcal{D}^*$ запись $|D|$ обозначает число компонентов в D (т.е. n). Если $D = (d_1, \ldots, d_n) \in \mathcal{D}^*$ и $1 \leq i \leq n$, то запись D_i обозначает i-компоненту этого кортежа (т.е. d_i). $\forall\, M \subseteq \mathcal{D}^*$, $\forall\, i \geq 1$ запись M_i обозначает множество i-х компонентов кортежей из M. Если кортеж $D = (d_1, \ldots, d_n) \in \mathcal{D}^*$ непуст, то записи $head(D)$ и $tail(D)$ обозначают значение $d_1 \in \mathcal{D}$ и кортеж $(d_2, \ldots, d_n) \in \mathcal{D}^*$ соответственно.

4.1.2. Связывания

Связыванием называется произвольная функция $\theta : \mathcal{X} \longrightarrow \mathcal{E}$.

Будем использовать следующие обозначения:

- множество всех связываний обозначается символом Θ,
- $\forall\, X \subseteq \mathcal{X}\quad \Theta(X) = \{\theta \in \Theta \mid \forall\, x \in \mathcal{X} \setminus X\ \ \theta(x) = x\}$,
- $\forall\, \theta \in \Theta,\ \forall\, e \in \mathcal{E}$ запись e^θ обозначает терм, получаемый из e заменой $\forall\, x \in \mathcal{X}_e$ каждого вхождения x в e на терм $\theta(x)$,
- $\forall\, \theta, \theta' \in \Theta$ запись $\theta\theta'$ обозначает связывание, определяемое следующим образом: $\forall\, x \in \mathcal{X}\ \ (\theta\theta')(x) = (x^\theta)^{\theta'}$.

4.2. Последовательные процессы

В этом пункте определяется понятие последовательного процесса. Данное понятие является моделью вычислительного процесса, порождаемого параллельной программой на каком-либо узле МПВС.

4.2.1. Действия

Элементарные действия (ЭД) – это записи следующих видов:

$$c!e, \quad c?e, \quad e := e', \quad [\![\varphi]\!], \quad \text{где } c \in \mathcal{C},\ e, e' \in \mathcal{E}, \tau_e = \tau_{e'}, \varphi \in \mathcal{B},$$

которые называются **посылкой** сообщения e в канал c, **приемом** сообщения e из канала c, **присваиванием** и **условным переходом** соответственно.

Действие – это конечная последовательность ЭД, в которой имеется не более одной посылки или приема. Действие называется **посылкой** или **приемом**, если одно из входящих в него ЭД – посылка или прием, соответственно. Действие, не являющееся посылкой или приемом, называется **внутренним действием**. Каждое ЭД можно рассматривать как действие, состоящее из одного этого ЭД.

Множество всех действий обозначается символом $\mathcal{A}$. $\forall\, \alpha \in \mathcal{A}$ множество всех переменных, входящих в α, обозначается записью $\mathcal{X}_\alpha$.

Если $\theta \in \Theta$ и $\alpha \in \mathcal{A}$, то запись α^θ обозначает действие, получаемое из α заменой каждой переменной x в α на терм x^θ.

4.2.2. Понятие последовательного процесса

Последовательный процесс (ПП) – это тройка (P, X, φ), компоненты которой имеют следующий смысл:

- P – граф с выделенной вершиной P^0 (называемой **начальной вершиной**), каждому ребру которого сопоставлена метка $\alpha \in \mathcal{A}$,
- $X \subseteq \mathcal{X}$ – множество **входных переменных** ПП P, и
- $\varphi \in \mathcal{B}$ – **начальное условие** ПП P.

Для каждого ПП (P, X, φ)

- данный ПП может сокращенно обозначаться тем же символом P, что и соответствующий ему граф, множество вершин графа P также обозначается символом P,
- X_P и φ_P обозначают вторую и третью компоненту P соответственно, $\mathcal{X}_P$ обозначает множество всех переменных, входящих в P,
- $\hat{X}_P$ обозначает множество $\mathcal{X}_P \setminus X_P$ **внутренних** переменных,
- $\mathcal{A}_P$ обозначает совокупность меток всех ребер графа P,
- $P^{v \to v'}$ обозначает произвольное ребро графа P из v в v'.

ПП является формальным описанием поведения динамической системы, работа которой заключается в последовательном выполнении действий, связанных с посылкой сообщений в каналы или приемом сообщений из каналов, а также с изменением значений внутренних переменных.

Будем предполагать, что для каждого ПП P

- каждая вершина графа P является элементом множества $\mathcal{D}$,
- P содержит внутреннюю переменную at_P, и для каждого ребра графа P, если v и v' – начало и конец соответственно этого ребра, то первое ЭД в метке этого ребра имеет вид $[\![at_P = v]\!]$, а последнее – $at_P := v'$, эти ЭД не будут указываться явно.

4.2.3. Состояние последовательного процесса

Состояние ПП P – это пара

$$s = (\theta^s, \{[c]^s \mid c \in \mathcal{D}_{\mathrm{C}}\}), \tag{6}$$

компоненты которой интерпретируются следующим образом:

- $\theta^s \in \Theta(\mathcal{X}_P)$ – связывание в s, причем $\forall\, x \in \mathcal{X}_P \;\; \mathcal{X}_{x^{\theta^s}} = \emptyset$,
- $\forall\, c \in \mathcal{D}_{\mathrm{C}} \;\; [c]^s \in \mathcal{D}^*$ – очередь непрочитанных значений в канале c в состоянии s ($[c]^s$ называется **содержимым** канала c в s).

Множество всех состояний ПП P обозначается записью Σ_P.

Для каждого состояния $s \in \Sigma_P$ и каждого терма $e \in \mathcal{E}(\mathcal{X}_P)$ будем обозначать значение e^{θ^s} записью e^s.

Состояние ПП P называется **начальным** (и обозначается 0_P), если оно имеет вид $(\theta, \{\varepsilon \mid c \in \mathcal{D}_{\mathrm{C}}\})$, где $\varphi_P^\theta = 1$ и $at_P^s = P^0$.

Состояние ПП P называется **терминальным**, если из вершины at_P^s не выходит ни одного ребра.

4.2.4. Переходы в последовательных процессах

В этом пункте мы определяем понятие перехода в ПП P, соответствующего какому-либо действию $\alpha \in \mathcal{A}_P$. Этот переход понимается как пара s, s' состояний из Σ_P, связь между которыми можно понимать следующим образом: если ПП P в текущий момент времени находился в состоянии s, то после последовательного выполнения ЭД, входящих в α, новым состоянием ПП P является состояние s'.

Будем обозначать записью $s \xrightarrow{\alpha} s'$ утверждение о том, что пара s, s' является переходом, соответствующим действию α, и определим это утверждение следующим образом: если действие $\alpha \in \mathcal{A}_P$

является последовательностью ЭД вида $\alpha_1 \dots \alpha_n$, то утверждение $s \xrightarrow{\alpha} s'$ верно, если

$$\exists\, s_1, \dots, s_{n-1} \in \Sigma_P \,:\, s \xrightarrow{\alpha_1} s_1,\ s_1 \xrightarrow{\alpha_2} s_2,\ \dots,\ s_{n-1} \xrightarrow{\alpha_n} s',$$

где утверждение $s \xrightarrow{\alpha} s'$ в том случае, когда α – ЭД, определяется отдельно для каждого возможного вида ЭД α (после формального определения данного утверждения для каждого конкретного вида α мы неформально интерпретируем изменение состояния в результате этого перехода):

(a) если $\alpha = c!e$, то $\theta^{s'} = \theta^s$, и

$$[c^s]^{s'} = ([c^s]^s, e^s), \quad \forall\, c' \in \mathcal{D}_{\mathrm{C}} \setminus \{c^s\} \quad [c']^{s'} = [c']^s,$$

в данном случае P посылает в канал c^s значение e^s, после чего

- связывание переменных ПП P не изменилось, и содержимое всех каналов, кроме канала c^s, также не изменилось,
- содержимое канала c^s увеличилось путем добавления к очереди $[c^s]^s$ значения e^s,

(b) если $\alpha = c?e$, то $[c^s]^s \neq \emptyset$ и

$$\begin{gathered}\exists\, \theta \in \Theta(\hat{X}_P) \,:\, (e^\theta)^s = head([c^s]^s), \theta^{s'} = \theta\theta^s,\\ [c^s]^{s'} = tail([c^s]^s), \quad \forall\, c' \in \mathcal{D}_{\mathrm{C}} \setminus \{c^s\} \quad [c']^{s'} = [c']^s\end{gathered}$$

в данном случае P принимает из канала c^s значение $head([c^s]^s)$ и изменяет текущее связывание так, чтобы значение терма e при новом связывании совпадало с принятым значением, после чего

- содержимое канала c^s стало $tail([c^s]^s)$,
- содержимое остальных каналов не изменилось,

(c) если $\alpha = (e := e')$, то

$$\begin{gathered}\exists\, \theta \in \Theta(\hat{X}_P) \,:\, (e^\theta)^s = (e')^s, \theta^{s'} = \theta\theta^s,\\ \forall\, c \in \mathcal{D}_{\mathrm{C}} \quad [c]^{s'} = [c]^s,\end{gathered} \tag{7}$$

в данном случае P изменяет текущее связывание так, чтобы значение терма e при новом связывании совпадало бы со значением терма e' при старом связывании, содержимое каналов не изменяется,

(d) если $\alpha = [\![\varphi]\!]$, то $\varphi^s = 1$, $\theta^{s'} = \theta^s$, и верно (7),
в данном случае связывание и содержимое каналов не изменяется.

Пустым переходом в ПП P называется произвольная пара состояний $s, s' \in \Sigma_P$, такая, что $\theta^s = \theta^{s'}$. Пустые переходы обозначаются записями вида $s \longrightarrow s'$.

Если пара s, s' является переходом в ПП, то s называется началом этого перехода, а s' – его концом.

4.2.5. Редукция графов последовательных процессов

Пусть заданы ПП P и вершина $v \in P$, не являющаяся начальной. Если множества всех ребер в P с концом в v и с началом в v имеют вид

$$\{v_i \xrightarrow{\alpha_i} v \mid i = 1, \dots, n\} \quad \text{и} \quad \{v \xrightarrow{\alpha'_i} v'_i \mid i = 1, \dots, n'\},$$

соответственно, где v отличается от всех вершин v_i и v'_i, и либо все действия $\alpha_1, \dots, \alpha_n$ внутренние, либо все действия $\alpha'_1, \dots, \alpha'_{n'}$ внутренние, то к данному графу можно применить операцию **редукции**, которая заключается в преобразовании данного графа путем

- удаления вершины v и связанных с нею ребер, и
- добавления ребер вида $v_i \xrightarrow{\alpha_i\alpha'_{i'}} v'_{i'}$, где $i = 1, \dots, n$, $i' = 1, \dots, n'$, и $\alpha_i\alpha'_{i'}$ – конкатенация последовательностей α_i и $\alpha'_{i'}$.

4.2.6. Переименования

Переименованием называется произвольная инъективная функция вида $\eta : X \longrightarrow X'$, где $X, X' \subseteq \mathcal{X}$.

Для каждого переименования $\eta : X \longrightarrow X'$, каждого $e \in \mathcal{E}$ и каждого ПП P записи e^η и P^η обозначают терм или ПП соответственно, получаемые из e или P заменой $\forall\, x \in X$ каждого вхождения x на $\eta(x)$.

Если P – ПП, и η – переименование вида $\eta : \hat{X}_P \longrightarrow \mathcal{X} \setminus X_P$, то будем рассматривать ПП P и P^η как равные.

4.3. Распределенные процессы

В этом пункте вводится понятие распределенного процесса, которое можно использовать для моделирования параллельных программ.

4.3.1. Понятие распределенного процесса

Распределенный процесс (РП) – это семейство ПП

$$\mathcal{P} = \{P_i \mid i \in I\}, \tag{8}$$

где компоненты семейства $\{\hat{X}_{P_i} \mid i \in I\}$ дизъюнктны и не пересекаются с множеством $X_\mathcal{P} \stackrel{\text{def}}{=} \bigcup_{i \in I} X_{P_i}$ (если это условие не выполняется, то заменим каждый ПП P_i на равный ему, в смысле, указанном в конце пункта 4.2.6, так, чтобы это условие выполнялось). Ниже будем считать, что данное условие всегда выполнено, даже если внутренние переменные в ПП P_i и $P_{i'}$ из $\mathcal{P}$, где $i \neq i'$, имеют одинаковые обозначения

Запись $\mathcal{X}_\mathcal{P}$ обозначает множество всех переменных, входящих в $\mathcal{P}$.

4.3.2. Понятие состояния распределенного процесса

Состоянием РП $\mathcal{P} = \{P_i \mid i \in I\}$ называется пара

$$s = (\theta^s, \{c^s \mid c \in \mathcal{D}_\mathrm{C}\}), \tag{9}$$

где $\theta^s \in \Theta(\mathcal{X}_\mathcal{P})$, и $\forall\, i \in I$

$$s_i \stackrel{\text{def}}{=} (\theta^s_i, \{c^s \mid c \in \mathcal{D}_\mathrm{C}\}) \in \Sigma_{P_i}$$

где $\theta^s_i \in \Theta(\mathcal{X}_{P_i})$, $\forall\, x \in \mathcal{X}_{P_i} \;\; x^{\theta^s_i} = x^{\theta^s}$.

Множество всех состояний РП $\mathcal{P}$ обозначается записью $\Sigma_\mathcal{P}$.

$\forall\, s \in \Sigma_\mathcal{P}$, $\forall\, e \in e \in \mathcal{E}(\mathcal{X}_\mathcal{P})$ будем обозначать значение e^{θ^s} записью e^s.

Состояние $s \in \Sigma_\mathcal{P}$ называется

- **начальным** (и обозначается $0_\mathcal{P}$), если $\forall\, i \in I \;\; s_i = 0_{P_i}$,
- **терминальным**, если $\forall\, i \in I$ состояние s_i терминально,
- **тупиковым**, если оно нетерминально, и $\forall\, i \in I$ не существует непустого перехода ПП P_i с началом s_i.

4.3.3. Переходы в распределенных процессах

Пусть заданы РП $\mathcal{P} = \{P_i \mid i \in I\}$, индекс $i \in I$ и действие $\alpha \in \mathcal{A}_{P_i}$.

Переходом в РП $\mathcal{P}$, соответствующим действию α ПП P_i, называется произвольная пара состояний $s, s' \in \Sigma_\mathcal{P}$, такая, что

$$s_i \stackrel{\alpha}{\longrightarrow} s'_i, \;\; \forall\, i' \in I \setminus \{i\} \;\; s_{i'} \longrightarrow s'_{i'}. \tag{10}$$

Будем обозначать свойство (10) записью $P_i^{\upsilon \to \upsilon'} : s \longrightarrow s'$, где υ, υ' – начало и конец соответственно ребра графа P_i с меткой α. Если пара s, s' является переходом в РП $\mathcal{P}$, то будем обозначать это записью $s \longrightarrow s'$.

Связь между состояниями $s, s' \in \Sigma_{\mathcal{P}}$, удовлетворяющими свойству (10), можно интерпретировать следующим образом: если РП $\mathcal{P}$ в текущий момент времени находился в состоянии s, и, начиная с этого момента, ПП P_i последовательно выполнял ЭД, входящие в α, а остальные ПП из $\mathcal{P}$ в течение всего этого времени не выполняли никаких действий, то после завершения выполнения последовательности ЭД, входящих в α, новым состоянием РП $\mathcal{P}$ является состояние s'.

Множество $\Sigma_{\mathcal{P}}$ можно рассматривать как граф, в котором существует ребро из s в s' с меткой α_{P_i} тогда и только тогда, когда верно (10).

Выполнение РП $\mathcal{P}$ представляет собой последовательность состояний $s_0, s_1, \ldots$ такой, что $s_0 = 0_{\mathcal{P}}$, и каждая пара s_i, s_{i+1} соседних состояний в этой последовательности является переходом в $\mathcal{P}$.

Состояние s РП $\mathcal{P}$ называется **достижимым**, если существует путь из $0_{\mathcal{P}}$ в s. Ниже $\Sigma_{\mathcal{P}}$ обозначает множество достижимых состояний $\mathcal{P}$.

4.4. Метод построения модели MPI-программы в виде распределенного процесса

Будем рассматривать только такие MPI-программы, которые содержат

- операторы присваивания, условного перехода, цикла,
- посылки и приема сообщений, определенные в пункте 2.2, и
- служебные MPI-функции (`MPI_Init` и т.п.), упомянутые в программе из пункта 3.2.

Если с MPI-программой П связаны множество $X_{\text{П}}$ входных переменных и начальное условие $\varphi_{\text{П}} \in \mathcal{B}$, то РП $\mathcal{P}_{\text{П}}$, являющийся моделью данной программы, имеет вид

$$\mathcal{P}_{\text{П}} = \{P_i \mid i = 0, 1, \ldots\}, \tag{11}$$

где $\forall\, i = 0, 1, \ldots$ ПП P_i строится следующим образом.

- Множество $\mathcal{X}_{P_i}$ состоит из переменных, входящих в $X_{\text{П}}$, и i-х копий внутренних переменных, входящих в П, $X_{P_i} = X_{\text{П}}$, $\varphi_{P_i} = \varphi_{\text{П}}$.
- Граф P_i строится путем
  - замены в программе П переменной, являющейся вторым аргументом функции `MPI_Comm_rank` (в MPI-программе, приведенной в пункте 3.2, это переменная `rank`) на константу i,
  - удаления неисполняемых частей получившейся программы, и
  - преобразования получившейся программы в графовую форму, аналогично тому, как по программе в операторной форме строится представление этой программы в виде блок-схемы (с тем отличием, что в блок-схемах действия связаны с вершинами, а в нашей модели они связаны с ребрами).

    Присваивания, условные операторы, операторы цикла преобразуются в действия ПП P_i стандартным образом, эти преобразования м.б. поняты из рассматриваемого ниже примера построения модели MPI-программы из пункта 3.2.

Используемые в П функции передачи сообщений представляются в графе P_i следующими действиями:

- функция (1) представляется действием $c_r!(e, i, l)$, где
  - e – терм, значение которого должно быть равно содержимому участка памяти, пересылаемому этой функцией,
  - r и l – соответствующие аргументы функции (1) (номер получателя и тег, соответственно),
- функция (2) представляется в P_i действием $c_i?(e, s, l)$, где
  - e – терм, значение которого после выполнения данного действия должно быть равно принятому сообщению,
  - s и l – новые переменные,

  и если последний аргумент в функции (2) имеет имя q, то выражения в P_i вида q`.MPI_SOURCE` и q`.MPI_TAG` заменяются на s и l, соответственно,

- представление функции (3) зависит от i:
  - при $i = 0$ функция (3) представляется действием $\circ!e$, где e – терм, значение которого должно быть равно содержимому участка памяти, пересылаемому этой функцией,
  - при $i \neq 0$ функция (3) представляется действием $\circ?e$, где e – терм, значение которого после выполнения данного действия должно быть равно принятому сообщению.

К построенному графу P_i можно применить операцию редукции, описанную в пункте 4.2.5.

4.5. Вспомогательные переменные

Для облегчения анализа РП $\mathcal{P}_\Pi$ можно добавить к действиям ПП, входящим в этот РП, присваивания вида $\iota := e$, в которых

- ι – новая переменная (называемая **вспомогательной** переменной),
- $e \in \mathcal{E}(\mathcal{X}_{\mathcal{P}_\Pi} \sqcup \mathcal{I})$, где $\mathcal{I}$ – множество вспомогательных переменных.

Присваивания указанного выше вида $\iota := e$ не являются реально выполняемыми действиями, они предназначены лишь для выражения зависимостей между значениями переменных во время выполнения РП. В некоторых случаях использование вспомогательных переменных позволяет компактно выразить свойства анализируемого РП и существенно упростить процедуру его анализа.

Последовательные процессы, получаемые из тех ПП, которые входят в РП $\mathcal{P}_\Pi$, добавлением присваиваний вида $\iota := e$, где ι – вспомогательная переменная, будем называть **дополненными** ПП.

5. Моделирование и верификация MPI-программы умножения матриц

В этом пункте мы рассмотрим пример применения описанных выше понятий для моделирования и верификации MPI-программы умножения матриц, представленной в пункте 3.1.

5.1. Построение модели MPI-программы умножения матриц

В этом пункте мы определяем РП $\mathcal{P}_\Pi = \{P_i \mid i = 0, 1, \ldots\}$, моделирующий MPI-программу П умножения матриц в пункте 3.1. Входными переменными П являются A, B (матрицы-сомножители), N (число строк в A), `nprocs`.

5.1.1. Упрощения

$\forall\, i \geq 0$ при построении ПП P_i используются следующие упрощения:

- оператор ввода матриц-сомножителей `input(a, b)` в строке 15 выполняется один раз в начальный момент выполнения процесса с рангом 0, поэтому в модели ПП P_0 можно опустить действия, соответствующие этому оператору, считая, что X_{P_0} содержит переменные A и B, значения которых равны матрицам-сомножителям,
- операторы КПС в P_0 в строках 16-17 и в P_i ($\forall\, i \geq 1$) в строках 44-45 выполняются один раз, в начальный момент выполнения этих ПП, поэтому можно заменить соответствующие действия вида $\circ!e$ и $\circ?e$ на предположение о том, что B входит в X_{P_i} ($\forall\, i \geq 1$).

Для сокращения обозначений в излагаемых ниже ПП вместо входной переменной `nprocs` будем использовать входную переменную n, значение которой равно `nprocs` – 1.

5.1.2. Последовательный процесс P_0

Для построения ПП P_0 используется только часть MPI-программы, расположенная в строках 12-39.

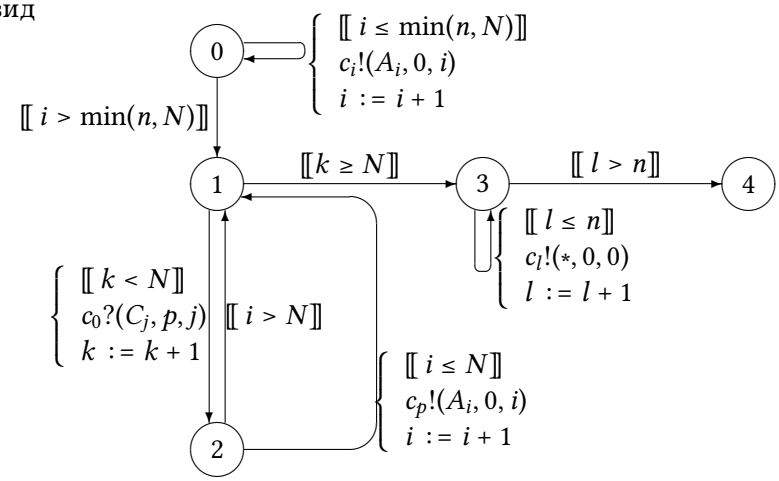
В ПП P_0 A и C являются именами массивов, компоненты которых являются строками соответствующих матриц и индексированы числами $1, \ldots, N$, i–е компоненты этих массивов обозначаются A_i и C_i.

P_0 имеет следующие переменные:

$$X_{P_0} = \{A, B, N, n\},\ \hat{X}_{P_0} = \{C, i, j, k, l, p\}.$$

Начальное условие: $\varphi_{P_0} = (N \geq 1) \wedge (i = 1) \wedge (k = 0) \wedge (l = 1)$.

ПП P_0 имеет вид



(12)

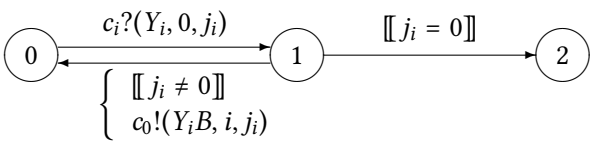
В ПП (12)

- – ребро $P_0^{0\to 0}$ соответствует циклу в строках 18-22 программы, переменная i в метке этого ребра соответствует выражению `count + 1` в программе,
  – сообщение, посылаемое функцией `MPI_Send` в строках 21-22 программы, представлено тройкой $(A_i, 0, i)$, третья компонента которой – тег этого сообщения (т.е. номер соответствующей строки в матрице A),
  – ребро $P_0^{0\to 1}$ соответствует выходу из этого цикла,
- – рёбра $P_0^{1\to 2}$ и $P_0^{2\to 1}$ соответствуют циклу в строках 23-34 программы,
  – переменная k соответствует переменной `i` в этом цикле,
  – оператор `MPI_Recv` в строках 24-26 и цикл в строках 27-28 программы заменены на единое действие: прием сообщения и его запись в соответствующую строку матрицы C,
  – ребро $P_0^{1\to 3}$ соответствует выходу из этого цикла,
- – ребра $P_0^{3\to 3}$ и $P_0^{3\to 4}$ соответствуют циклу в строках 35-37,
  – символ $*$ в метке ребра $P_0^{3\to 3}$ изображает пустую строку.

5.1.3. Последовательный процесс P_i для $i > 0$

Для построения ПП P_i, где $i \geq 1$, используется только часть MPI-программы, расположенная в строках 42-55.

P_i имеет следующие переменные: $X_{P_i} = \{B\}, \hat{X}_{P_i} = \{Y_i, j_i\}$, где значением B является вторая матрица-сомножитель, и значениями Y_i – строки действительных чисел. ПП P_i имеет следующий вид:



(13)

5.2. Верификация распределенного процесса, моделирующего MPI-программу умножения матриц

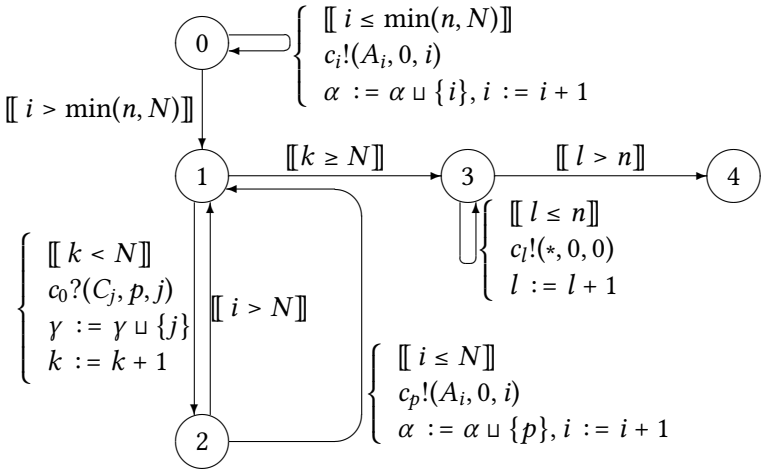
5.2.1. Вспомогательные переменные

Для доказательства утверждения о том, что определенный в пункте 5.1 РП $\mathcal{P}_\Pi$, моделирующий MPI-программу П умножения матриц в пункте 3.1, удовлетворяет своей спецификации, выражаемой свойством (4), мы введем вспомогательные переменные α, β, γ (и связанные с ними действия, не влияющие на выполнение РП), значения которых будут иметь следующий смысл: $\forall\, s \in \Sigma_{P_\Pi}$

- $\alpha^s \subseteq \{1, \ldots, n\}$, α^s состоит из номеров каналов из $\{c_1, \ldots, c_n\}$ с непустым содержимым в состоянии s,
- $\beta^s \subseteq \{1, \ldots, N\}$, β^s состоит из номеров строк матрицы A, для которых в состоянии s вычисляется их произведение на B,
- $\gamma^s \subseteq \{1, \ldots, N\}$, γ^s равно множеству номеров всех строк, которые P_0 записал в C до момента прихода РП $\mathcal{P}_\Pi$ в состояние s.

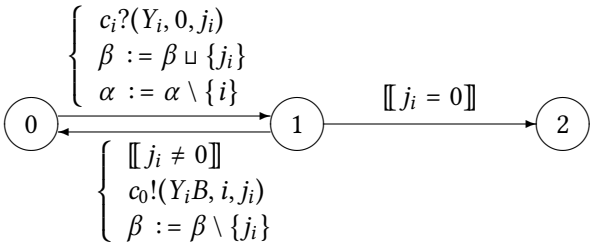
Начальные значения α, β и γ равны $\emptyset$.

Дополненный ПП P_0 имеет вид



(14)

$\forall\, i = 1, \ldots, n$ дополненный ПП P_i имеет вид



(15)

Использование в (14) и (15) символа $\sqcup$ для операций объединения множеств выражает утверждение (вытекающего из нижеследующей теоремы 1) о том, что всякий раз при выполнении этих операций их аргументы в действительности являются дизъюнктными множествами.

5.2.2. Теоремы, обосновывающие корректность $\mathcal{P}_\Pi$

Теорема 1.

$\forall\, s \in \Sigma_{\mathcal{P}_\Pi}$, если $at^s_{P_0} \neq 3, 4$, то верны следующие утверждения:

1. $\alpha^s \subseteq \{1, \ldots, i^s - 1\}$,
2. $i^s - 1 \leq N$
3. $|\gamma^s| = k^s \leq N$,
4. если $at^s_{P_0} = 1$ и $k^s < N$, то $k^s < i^s - 1$,
5. $[c_0]^s_2 \cap \alpha^s = \emptyset$,
6. $\forall\, i = 1, \ldots, n$
   (a) $|[c_i]^s| = \begin{cases} 1, \text{ если } i \in \alpha^s, \text{ и} \\ 0 \text{ иначе,} \end{cases}$
   (b) $at^s_{P_i} = 1 \implies (i \notin \alpha^s) \wedge (j^s_i \notin \gamma^s)$
7. $at^s_{P_0} = 2 \implies p^s \notin \alpha^s, p^s \in \{1, \ldots, i^s - 1\}$,

8. $[c_1]_3^s \sqcup \ldots \sqcup [c_n]_3^s \sqcup \beta^s \sqcup [c_0]_3^s \sqcup \gamma^s = \{1, \ldots, i^s - 1\}$,
9. $\forall\, p \in \alpha^s \;\; [c_p]^s$ имеет вид $\{(A_i, 0, i)\}$, где $i \in \{1, \ldots, N\}$,
10. каждый элемент $[c_0]^s$ имеет вид $(A_i B, p, i)$, где $i \in \{1, \ldots, N\}$,
11. $\forall\, j \in \gamma^s \;\; C_j = A_j B$,
12. $k^s = N \implies \gamma^s = \{1, \ldots, N\}$.

Доказательство.

Истинность всех вышеперечисленных утверждений обосновывается индуктивно: все они верны в начальном состоянии $0_{\mathcal{P}_\Pi}$, и сохраняют свою истинность после каждого перехода $s \longrightarrow s'$ РП $\mathcal{P}_\Pi$, где $at_{P_0}^{s'} \neq 3, 4$. ■

Теорема 2.

В $\Sigma_{\mathcal{P}_\Pi}$ нет тупиковых состояний.

Доказательство.

Пусть $\Sigma_{\mathcal{P}_\Pi}$ содержит тупиковое состояние s. Нетрудно доказать, что $at_{P_0}^s$ не м.б. равно 0, 2, 3, и $\forall\, i = 1, \ldots, n$ $at_{P_i}^s$ не м.б. равно 1. Таким образом, для $at_{P_0}^s$ есть два возможных значения: 1 и 4.

1. Пусть $at_{P_0}^s = 1$. Из тупиковости s следует, что $k^s < N$, откуда опять используя тупиковость s получаем: $[c_0]^s = \varnothing$.

   Из утверждения 4 в теореме 1 следует, что $k^s < i^s - 1$, откуда на основании утверждения 8 теоремы 1 и равенства $[c_0]^s = \varnothing$ получаем:

   $$[c_1]_3^s \sqcup \ldots \sqcup [c_n]_3^s \sqcup \beta^s \neq \varnothing. \tag{16}$$

   Если $\exists\, i \in \{1, \ldots, n\}$: $[c_i]^s \neq \varnothing$, то из тупиковости s следует, что $at_{P_i}^s \neq 0$, поэтому $at_{P_i}^s = 2$, откуда нетрудно получить, что $at_{P_0}^s = 4$. Однако это возможно лишь в случае $k^s \geq N$, что противоречит установленному выше неравенству $k^s < N$.

   Поэтому $\forall\, i \in \{1, \ldots, n\}$ $[c_i]^s = \varnothing$, и из (16) следует, что $\beta \neq \varnothing$. Однако анализом ПП P_i ($i = 0, \ldots, n$) нетрудно установить, что это возможно только если $\exists\, i \in \{1, \ldots, n\}$: $at_{P_i}^s = 2$. Как было установлено в предыдущем абзаце, данное равенство невозможно.
2. Пусть $at_{P_0}^s = 4$. Тогда $k^s \geq N$, откуда следует, что $|\gamma^s| = k^s = N$.

   Пусть s' – первое состояние на пути π из $0_{\mathcal{P}_\Pi}$ в s, такое, что $k^{s'} = N$. Нетрудно видеть, что $at_{P_0}^{s'} = 2$. Из утверждений 2 и 8 теоремы 1 следует, что $i^{s'} - 1 = N$ и

   $$[c_0]^{s'} = [c_1]^{s'} = \ldots = [c_n]^{s'} = \beta^{s'} = \varnothing,$$
   $$at_{P_1}^{s'} = 0, \ldots, at_{P_n}^{s'} = 0.$$

   Единственный переход из s' соответствует действию $[\![i > N]\!]$ и имеет вид $P_0^{2 \to 1} : s' \longrightarrow s''$, единственный переход из s'' соответствует действию $[\![k \geq N]\!]$ и имеет вид $P_0^{1 \to 3} : s'' \longrightarrow s'''$. Нетрудно видеть, что все состояния в хвосте пути π, начинающемся с s''', не являются тупиковыми.

В обоих случаях мы получили противоречие, которое является следствием предположения о том, что в $\Sigma_{\mathcal{P}_\Pi}$ содержит тупиковое состояние. Следовательно, в $\Sigma_{\mathcal{P}_\Pi}$ нет тупиковых состояний. ■

Теорема 3.

Любое выполнение определенного выше РП $\mathcal{P}$ завершается после конечного числа шагов.

Доказательство.

Пусть существует бесконечное выполнение π РП $\mathcal{P}$.

Докажем, что число переходов в π, соответствующих действиям ПП P_0, конечно. Если бы число таких переходов было бесконечным, то

- в π нет состояний s, обладающих свойством $at^s_{P_0} = 3$,
- существует состояние $s_1 \in \pi$, такое, что $at^{s_1}_{P_0} = 1$,
- каждый переход в π, начиная с s_1, соответствующий какому-либо действию P_0, имеет вид либо $P_0^{1\to 2} : s \longrightarrow s'$, либо $P_0^{2\to 1} : s \longrightarrow s'$, и число переходов вида $P_0^{1\to 2} : s \longrightarrow s'$ бесконечно, что невозможно по причине того, что при каждом таком переходе увеличивается значение переменной k, которое, согласно утверждению 3 теоремы 1, ограничено сверху значением N.

Пусть $s' \in \pi$ – состояние, начиная с которого π не содержит переходов, соответствующих действиям P_0, и π' – часть пути π, начинающаяся с s'. Нетрудно видеть, что

$$\exists\, i \in \{1, \ldots, n\}\colon \pi' \text{ содержит бесконечно много переходов вида } P_i^{0\to 1} : s \longrightarrow s'. \tag{17}$$

Т. к. π' не содержит переходов, соответствующих действиям P_0, то значение $|[c_i]^s|$ не может увеличиваться, и согласно (17) оно бесконечно уменьшается, что невозможно. ■

Из доказанных теорем следует, что каждое выполнение РП $\mathcal{P}_\Pi$ является конечным и завершается в некотором терминальном состоянии s. Из $at^s_{P_0} = 4$ следует, что $|\gamma^s| = k^s = N$, откуда согласно утверждениям 11 и 12 теоремы 1 следует, что РП $\mathcal{P}_\Pi$ удовлетворяет спецификации, изложенной в пункте 3.4.

6. Заключение

В настоящей работе мы представили новую математическую модель параллельных программ, предложили подход к верификации таких программ и рассмотрели пример верификации MPI-программы умножения матриц на основе предложенной модели. Основное преимущество предложенного подхода – возможность его применения для программ, генерирующих неограниченный набор последовательных процессов.

Проблемы для дальнейших исследований, связанных с предлагаемой моделью, могут быть следующими:

- расширить предложенную модель, введя концепции для моделирования синхронной передачи сообщений и другие механизмы для организации параллельного выполнения,
- ввести язык спецификации свойств распределенных процессов, на котором свойства параллельных программ выражаются в терминах наблюдаемой эквивалентности, и разработать алгоритм для распознавания наблюдаемой эквивалентности распределенных процессов.

References

[1] G. J. Holzmann, *Design and Validation of Computer Protocols*. Prentice Hall, 1990.

[2] H. A. Lopez, E. R. Marques, F. Martins, N. Ng, C. Santos, V. T. Vasconcelos, and N. Yoshida, “Protocol-based verification of message-passing parallel programs”, in *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications - OOPSLA 2015*, 2015, pp. 280–298.

[3] I. Grudenic and N. Bogunovic, “Modeling and Verification of MPI Based Distributed Software”, in *Recent Advances in Parallel Virtual Machine and Message Passing Interface. EuroPVM/MPI 2006*, ser. LNCS, vol. 4192, Springer, 2006, pp. 123–132.

[4] A. Blass and Y. Gurevich, “Abstract State Machines Capture Parallel Algorithms”, in *ACM Transactions on Computational Logic*, 2003, pp. 578–651.

[5] S. F. Siegel, “Model Checking Nonblocking MPI Programs”, in *International Workshop on Verification, Model Checking, and Abstract Interpretation, VMCAI*, ser. LNCS, vol. 4349, Springer, 2007, pp. 44–58.

[6] S. F. Siegel, A. Mironova, G. S. Avrunin, and L. A. Clarke, “Combining symbolic execution with model checking to verify parallel numerical programs”, *ACM Transactions on Software Engineering and Methodology*, vol. 17, no. 2, pp. 1–34, 2008.

[7] S. Vakkalanka, G. Gopalakrishnan, and R. M. Kirby, “Dynamic Verification of MPI Programs with Reductions in Presence of Split Operations and Relaxed Orderings”, in *International Conference on Computer Aided Verification CAV 2008*, ser. LNCS, vol. 5123, Springer, 2008, pp. 66–79.

[8] G. Gopalakrishnan, R. M. Kirby, S. Siegel, R. Thakur, W. Gropp, E. Lusk, B. R. De Supinski, M. Schulz, and G. Bronevetsky, “Formal analysis of MPI-based parallel programs”, *Communications ACM*, vol. 54, no. 12, pp. 82–91, 2011.

[9] V. Forejt, S. Joshi, D. Kroening, G. Narayanaswamy, and S. Sharma, “Precise Predictive Analysis for Discovering Communication Deadlocks in MPI Programs”, *ACM Transactions on Programming Languages and Systems*, vol. 39, no. 4, pp. 1–27, 2017.

[10] Z. Luo, M. Zheng, and S. F. Siegel, “Verification of MPI programs using CIVL”, in *EuroMPI*, 2017, 6:1–6:11.

[11] W. Hong, Z. Chen, H. Yu, and J. Wang, “Evaluation of model checkers by verifying message passing programs”, in *Science China Information Sciences*, vol. 62, 2019.

[12] H. Yu, Z. Chen, X. Fu, J. Wang, Z. Su, J. Sun, C. Huang, and W. Dong, “Symbolic Verification of Message Passing Interface Programs”, in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, ICSE ’20*, 2020, pp. 1248–1260.