Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИУ «Информатика и системы управления»

КАФЕДРА ИУ-7 «Программное обеспечение ЭВМ и информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА №1

«Знакомство с языком Promela»

Группа: ИУ7-41М

Студент: Дубовицкая Ольга Николаевна

Дисциплина: Математические основы верификации ПО

Преподаватель: Кузнецова Ольга Владимировна

Москва, 2025 г.

Задание

Для небольшого фрагмента программы (10-15 строк кода, в которых есть изменение значений переменных) необходимо описать модель этой программы на Promela и изучить её (SPIN).

Отчёт должен содержать:

- фрагмент кода,
- описание модели,
- перечисление множества состояний и текстовое пояснение причин переходов между состояниями,
- граф переходов между состояниями модели,
- вывод по работе.

Фрагмент кода

Рассмотрим программу, реализующую рекурсивную функцию для вычисления факториала числа (Листинг 1).

Факториал числа n (обозначается как n!) – это произведение всех положительных целых чисел от 1 до n.

Основная логика программы реализована в рамках функции *factorial*, которая принимает целое число n и возвращает также целое число, соответствующее значению вычисляемого факториала.

Если n меньше или равно 1, функция возвратит 1 в соответствии со свойствами факториала.

В противном случае функция будет вызывать себя рекурсивно с аргументом n-1. Результат этого вызова сохранится в переменной result. После чего функция возвратит значение, соответствующее результату вычисления факториала.

Внутри функции *main*, точки входа в программу, будет происходить вычисление факториала 7! (в соответствии с определённой константой UPPER_LIMIT) с последующим выводом результата на экран.

Листинг 1

```
#include <iostream>

#define UPPER_LIMIT 7

int factorial(int n) {
    if (n <= 1) {
        return 1; // базовый случай

    } else {
        int result = factorial(n - 1); // рекурсивный вызов
        return n * result; // возвращаем результат
    }
}

int main() {
    int result = factorial(UPPER_LIMIT);
    std::cout << UPPER_LIMIT << "! = " << result << std::endl;
    return 0;
}
```

Описание модели

Рассматриваемая модель, представленная в Листинге 2, реализует рекурсивное вычисление факториала числа, где каждый вызов *factorial* происходит в отдельном процессе.

Взаимодействие между процессами осуществляется через каналы для передачи сообщений (это делает код более модульным и упрощает управление данными). Используются сообщения двух типов:

– BC (BASE_CASE) – базовый случай вычисления факториала (т.е. это вычисление 1!),

– RC (RECURSIVE_CASE) – рекурсивный случай вычисления факториала.

Каждое передаваемое сообщение содержит два поля:

– msg_type – тип сообщения (базовый или рекурсивный случай),

– msg_data – данные сообщения (в рассматриваемой модели – результат вычисления факториала).

Процесс *factorial* принимает два параметра – целое число n и родительский канал parent_channel, по которому будет отправляться результат вычисления факториала. Для передачи сообщений между дочерними

процессами создаётся канал child_channel. Оба этих канала позволяют одновременно содержать только одно сообщение.

Базовый случай: если n меньше или равно 1, создаётся сообщение с типом BC и значением 1, которое затем отправляется в родительский канал parent_channel.

Рекурсивный случай: если n больше или равно 2, будет запущен новый процесс factorial с аргументом n-1 и каналом child_channel. Затем ожидается получение сообщения от дочернего процесса по каналу child_channel. Результат вычисления факториала, полученный от дочернего процесса, будет сохранён в переменной result. После чего создаётся сообщение с типом RC и значением n * result, которое затем будет отправлено в родительский канал.

В блоке *init* будет создан канал child_channel для передачи сообщений и переменная для хранения результата вычисления факториала. Производится вычисление факториала 7! в соответствии с определённой константой UPPER_LIMIT. После запуска процесса factorial ожидается получение результата в сообщении из канала child_channel. В дальнейшем значение вычисленного интеграла будет выведено на экран.

Листинг 2

```
#define UPPER_LIMIT 7

mtype = { BC, RC };

typedef Message {
    mtype msg_type;
    int msg_data;
};

proctype factorial(int n; chan parent_channel){
    chan child_channel = [1] of { Message };
    Message msg;
    int result;

    if
    :: (n <= 1) ->
        msg.msg_type = BC;
        msg.msg_data = 1;

        parent_channel ! msg
```

```
    :: (n >= 2) ->
        run factorial(n-1, child_channel);
        child_channel ? msg;

        result = msg.msg_data;

        msg.msg_type = RC;
        msg.msg_data = n * result;

        parent_channel ! msg
    fi
}

init {
    chan child_channel = [1] of { Message };
    Message factorial_result;

    run factorial(UPPER_LIMIT, child_channel);
    child_channel ? factorial_result;
    printf("%d! = %d\n", UPPER_LIMIT, factorial_result.msg_data)
}
```

В общем виде обмен сообщениями между процессами приведён на рисунке 1. Листинг 3, который следует за рисунком, содержит последовательность действий процессов, выполняющих вычисление факториала.

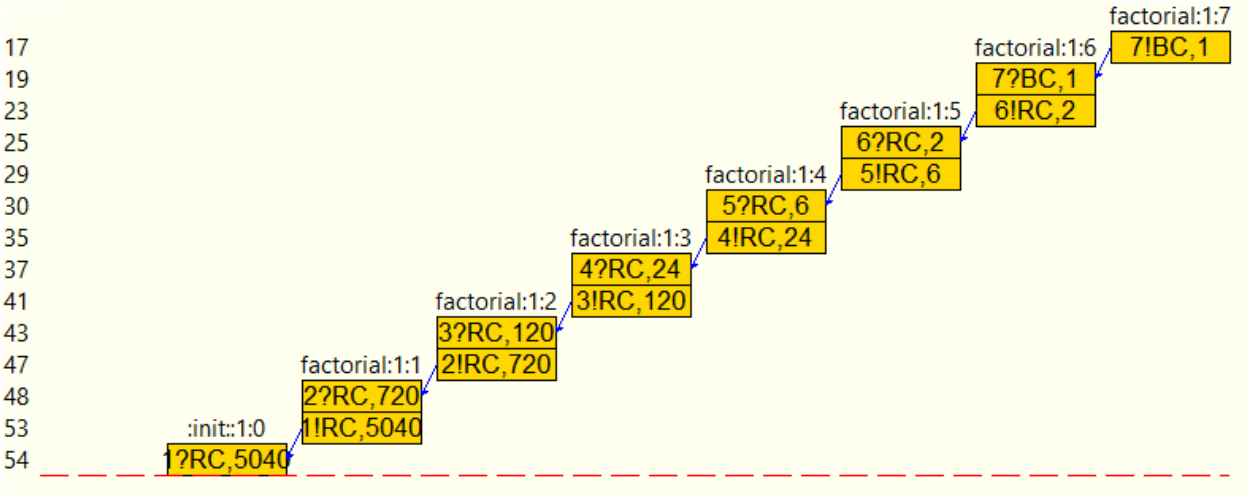

Рисунок 1. Обмен сообщениями между процессами

Листинг 3

```
  0:   proc  - (:root:) creates proc  0 (:init:)
Starting factorial with pid 1
  1:   proc  0 (:init::1) creates proc  1 (factorial)
  1:   proc  0 (:init::1) factorial.pml:39 (state 1)[(run
factorial(7,child_channel))]
  2:   proc  1 (factorial:1) factorial.pml:22 (state 5)    [((n>=2))]
Starting factorial with pid 2
```

```
  3:   proc  1 (factorial:1) creates proc  2 (factorial)
  3:   proc  1 (factorial:1) factorial.pml:23 (state 6)    [(run factorial((n-
1),child_channel))]
  4:   proc  2 (factorial:1) factorial.pml:22 (state 5)    [((n>=2))]
Starting factorial with pid 3
  5:   proc  2 (factorial:1) creates proc  3 (factorial)
  5:   proc  2 (factorial:1) factorial.pml:23 (state 6)    [(run factorial((n-
1),child_channel))]
  6:   proc  3 (factorial:1) factorial.pml:22 (state 5)    [((n>=2))]
Starting factorial with pid 4
  7:   proc  3 (factorial:1) creates proc  4 (factorial)
  7:   proc  3 (factorial:1) factorial.pml:23 (state 6)    [(run factorial((n-
1),child_channel))]
  8:   proc  4 (factorial:1) factorial.pml:22 (state 5)    [((n>=2))]
Starting factorial with pid 5
  9:   proc  4 (factorial:1) creates proc  5 (factorial)
  9:   proc  4 (factorial:1) factorial.pml:23 (state 6)    [(run factorial((n-
1),child_channel))]
 10:   proc  5 (factorial:1) factorial.pml:22 (state 5)    [((n>=2))]
Starting factorial with pid 6
 11:   proc  5 (factorial:1) creates proc  6 (factorial)
 11:   proc  5 (factorial:1) factorial.pml:23 (state 6)    [(run factorial((n-
1),child_channel))]
 12:   proc  6 (factorial:1) factorial.pml:22 (state 5)    [((n>=2))]
Starting factorial with pid 7
 13:   proc  6 (factorial:1) creates proc  7 (factorial)
 13:   proc  6 (factorial:1) factorial.pml:23 (state 6)    [(run factorial((n-
1),child_channel))]
 14:   proc  7 (factorial:1) factorial.pml:16 (state 1)    [((n<=1))]
 15:   proc  7 (factorial:1) factorial.pml:17 (state 2)    [msg.msg_type = BC]
 16:   proc  7 (factorial:1) factorial.pml:18 (state 3)    [msg.msg_data = 1]
 17:   proc  7 (factorial:1) factorial.pml:20 (state 4)
       [parent_channel!msg.msg_type,msg.msg_data]
 18:   proc  7 (factorial:1) terminates
 19:   proc  6 (factorial:1) factorial.pml:24 (state 7)
       [child_channel?msg.msg_type,msg.msg_data]
 20:   proc  6 (factorial:1) factorial.pml:26 (state 8)    [result = msg.msg_data]
 21:   proc  6 (factorial:1) factorial.pml:28 (state 9)    [msg.msg_type = RC]
 22:   proc  6 (factorial:1) factorial.pml:29 (state 10)   [msg.msg_data =
(n*result)]
 23:   proc  6 (factorial:1) factorial.pml:31 (state 11)
       [parent_channel!msg.msg_type,msg.msg_data]
 24:   proc  6 (factorial:1) terminates
 25:   proc  5 (factorial:1) factorial.pml:24 (state 7)
       [child_channel?msg.msg_type,msg.msg_data]
 26:   proc  5 (factorial:1) factorial.pml:26 (state 8)    [result = msg.msg_data]
 27:   proc  5 (factorial:1) factorial.pml:28 (state 9)    [msg.msg_type = RC]
 28:   proc  5 (factorial:1) factorial.pml:29 (state 10)   [msg.msg_data =
(n*result)]
 29:   proc  5 (factorial:1) factorial.pml:31 (state 11)
       [parent_channel!msg.msg_type,msg.msg_data]
 30:   proc  4 (factorial:1) factorial.pml:24 (state 7)
       [child_channel?msg.msg_type,msg.msg_data]
 31:   proc  4 (factorial:1) factorial.pml:26 (state 8)    [result = msg.msg_data]
 32:   proc  4 (factorial:1) factorial.pml:28 (state 9)    [msg.msg_type = RC]
 34:   proc  4 (factorial:1) factorial.pml:29 (state 10)   [msg.msg_data =
(n*result)]
 35:   proc  4 (factorial:1) factorial.pml:31 (state 11)
       [parent_channel!msg.msg_type,msg.msg_data]
 37:   proc  3 (factorial:1) factorial.pml:24 (state 7)
       [child_channel?msg.msg_type,msg.msg_data]
```

```
 37:   proc  5 (factorial:1) terminates
 37:   proc  4 (factorial:1) terminates
 38:   proc  3 (factorial:1) factorial.pml:26 (state 8)    [result = msg.msg_data]
 39:   proc  3 (factorial:1) factorial.pml:28 (state 9)    [msg.msg_type = RC]
 40:   proc  3 (factorial:1) factorial.pml:29 (state 10)   [msg.msg_data =
(n*result)]
 41:   proc  3 (factorial:1) factorial.pml:31 (state 11)
       [parent_channel!msg.msg_type,msg.msg_data]
 43:   proc  2 (factorial:1) factorial.pml:24 (state 7)
       [child_channel?msg.msg_type,msg.msg_data]
 43:   proc  3 (factorial:1) terminates
 44:   proc  2 (factorial:1) factorial.pml:26 (state 8)    [result = msg.msg_data]
 45:   proc  2 (factorial:1) factorial.pml:28 (state 9)    [msg.msg_type = RC]
 46:   proc  2 (factorial:1) factorial.pml:29 (state 10)   [msg.msg_data =
(n*result)]
 47:   proc  2 (factorial:1) factorial.pml:31 (state 11)
       [parent_channel!msg.msg_type,msg.msg_data]
 48:   proc  1 (factorial:1) factorial.pml:24 (state 7)
       [child_channel?msg.msg_type,msg.msg_data]
 50:   proc  1 (factorial:1) factorial.pml:26 (state 8)    [result = msg.msg_data]
 51:   proc  1 (factorial:1) factorial.pml:28 (state 9)    [msg.msg_type = RC]
 52:   proc  1 (factorial:1) factorial.pml:29 (state 10)   [msg.msg_data =
(n*result)]
 52:   proc  2 (factorial:1) terminates
 53:   proc  1 (factorial:1) factorial.pml:31 (state 11)
       [parent_channel!msg.msg_type,msg.msg_data]
 54:   proc  0 (:init::1) factorial.pml:40 (state 2)
       [child_channel?factorial_result.msg_type,factorial_result.msg_data]
 55:   proc  0 (:init::1) factorial.pml:41 (state 3)[printf('%d! =
%d\\n',7,factorial_result.msg_data)]
 56:   proc  1 (factorial:1) terminates
 56:   proc  0 (:init::1) terminates
8 processes created
```

Каждый процесс в Листинге 3 обозначается как *proc X*, где *X* – это идентификатор процесса. Например, *proc 0* – это основной процесс, который создаёт другие дочерние процессы.

Каждая строка Листинга 3 содержит информацию о состоянии того или иного процесса, включая его идентификатор, имя (например, *factorial:1*), номер строки кода (например, *factorial.pml:39*), текущее состояние (например, *(state 1)*) и условия выполнения (например, *[(run factorial(7, child_channel))]*).

Количество созданных процессов (в данном случае 8) соответствует количеству рекурсивных вызовов функции для вычисления факториала.

Когда один процесс создаёт другой новый, это обозначается строкой вида «*proc X creates proc Y*», где X – родительский процесс, а Y – новый дочерний процесс. Процессы обмениваются сообщениями через каналы, что позволяет им взаимодействовать и передавать данные друг другу.

Рассмотрим основные моменты вывода, представленного в Листинге 3.

1) **Инициализация**

– «*0: proc – (:root:) creates proc 0 (:init:) Starting factorial with pid 1*» – корневой процесс создаёт начальный процесс 0 с PID 1 для вычисления факториала,

– «*1: proc 0 (:init::1) creates proc 1 (factorial)*» – процесс 0 с PID 1 создаёт процесс 1 с PID 2 для вычисления факториала

2) **Рекурсия**

– «*3: proc 1 (factorial:1) factorial.pml:23 (state 6) [(run factorial ((n-1), child_channel))]*» и «*4: proc 2 (factorial:1) factorial.pml:22 (state 5) [((n>=2))] Starting factorial with pid 3*» – процесс 1 с PID 2 создаёт и запускает процесс 2 с PID 3 для вычисления *factorial(6)* и так далее; подобные вызовы будут рекурсивно повторяться до тех пор, пока не окажется достигнуто условие выхода из рекурсии, т.е. $n \leq 1$,

– каждый новый процесс получает свой уникальный PID и создаёт следующий процесс, пока не дойдёт до *factorial(1)*.

3) **Базовый случай**

Как только достигается *factorial(1)*, выполняются следующие действия:

– «*14: proc 7 (factorial:1) factorial.pml:16 (state 1) [((n<=1))]*» – процесс 7 проверяет, что $n \leq 1$,

– «*17: proc 7 (factorial:1) factorial.pml:20 (state 4) [parent_channel ! msg.msg_type, msg.msg_data]*» – происходит отправка процессом 7 сообщения с результатом 1 обратно своему родительскому процессу.

4) **Возврат результатов**

- каждый дочерний процесс после получения результата от своего дочернего процесса начинает вычислять свой результат; так, например, proc 6 получает результат от proc 7, умножает его на n (в данном случае на 2) и отправляет процессу 5 (*см. строчки 19-23 Листинга 3*),
- этот процесс продолжается вверх по цепочке до тех пор, пока не будет возвращён окончательный результат в корневой процесс,
- после того как все процессы завершили свои расчёты и отправили результаты обратно своим родителям, они завершаются; например, на завершение процесса 7 указывает строчка: «*18: proc 7 (factorial:1) terminates*».

Множества состояний и графы переходов между состояниями

Процесс *init*

- init – начальное состояние,
- S1 – инициализация необходимых переменных и каналов для запуска первого процесса вычисления факториала,
- S2 – запуск процесса factorial с аргументом 7 и каналом child_channel для получения сообщения с результатом вычислений,
- S3 – ожидание результата из канала child_channel,
- S4 – вывод полученного результата на экран,
- S0 – терминальное состояние.

Процесс *factorial*

- factorial – начальное состояние,
- S12 – проверка параметра n,
- S2 – параметр n меньше или равен 1,
- S3 – инициализация типа передаваемого по родительскому каналу parent_channel сообщения msg как базового случая (BC),
- S4 – инициализация передаваемого в сообщении msg значения как 1,
- S6 – параметр n больше или равен 2,

- S7 – создание дочернего процесса,
- S8 – ожидание результата от дочернего процесса из канала child_channel,
- S9 – сохранение полученного от дочернего процесса значения в переменную result,
- S10 – инициализация типа передаваемого по родительскому каналу parent_channel сообщения msg как рекурсивного случая (RC),
- S11 – инициализация передаваемого в сообщении msg значения как n * result, где result -–это результат полученный от дочернего процесса,
- S14 – отправка сообщений с результатами родительскому процессу parent_channel,
- S0 – терминальное состояние.

На рисунках 2 и 3 представлены графы переходов между состояниями процессов *init* и *factorial* соответственно.

Красным цветом на графах отмечены те рёбра, которые отвечают за наиболее важные переходы.

В графе *init* это вызов процесса factorial и получение результатов из канала child_channel. Эти действия являются основными шагами при вычислении факториала.

В графе *factorial* соответственно выделены такие ключевые шаги, как рекурсивное создание дочерних процессов и отправка результатов обратно родительскому каналу.

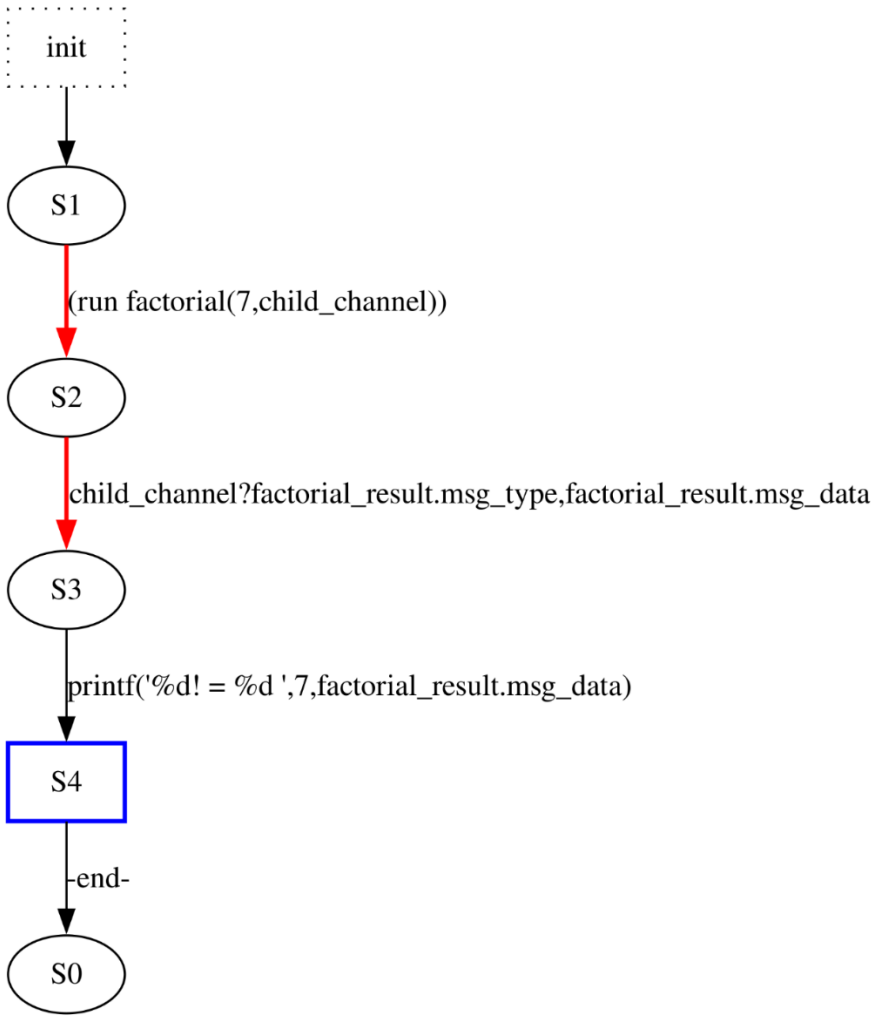

Рисунок 2. Граф переходов между состояниями процесса *init*

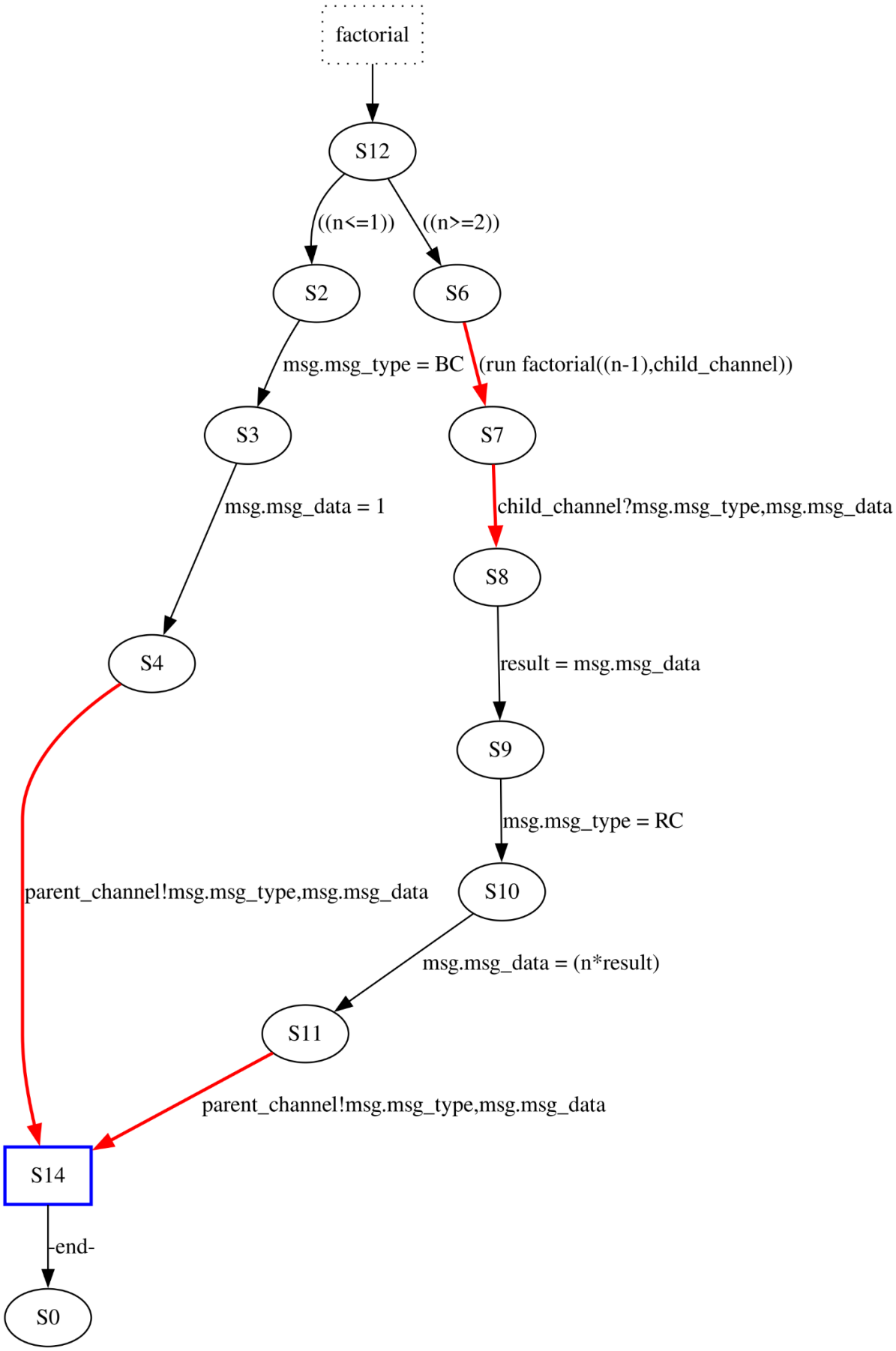

Рисунок 3. Граф переходов между состояниями процесса *factorial*

Выводы

В результате выполнения лабораторной работы были изучены базовые возможности языка Promela (Process Meta Language), на котором была описана модель программы для вычисления факториала числа.

Были освоены базовые возможности верификатора SPIN (Simple Promela Interpreter), с помощью которого оказались получены графы переходов между состояниями процессов программы. Была рассмотрена и описана последовательность обмена сообщениями между процессами и в целом последовательность их действий при работе с рекурсивной функцией, вычисляющей факториал.