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 «Информатика и системы управления (ИУ)»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии (ИУ7)»

ОТЧЕТ

Лабораторная работа №1

по курсу «Математические основы верификации ПО»

на тему: «Знакомство с языком Promela»

Студент ИУ7-42М (Группа) _____ (Подпись, дата) К.Э. Ковалец (И. О. Фамилия)

Преподаватель _____ (Подпись, дата) О.В. Кузнецова (И. О. Фамилия)

2025 г.

1 Выполнение лабораторной работы

Promela (Process Meta Language) — это язык программирования, разработанный для описания и верификации распределённых и параллельных систем. Он используется в связке с инструментом SPIN (Simple Promela Interpreter) для моделирования, проверки корректности и поиска ошибок в системах с параллельными процессами. SPIN позволяет выполнять формальную верификацию моделей, написанных на языке Promela, и проверять их на наличие различных типов ошибок, таких как взаимоблокировки, гонки данных и другие проблемы, связанные с параллелизмом.

1.1 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.

1.2 Фрагмент кода

Код программы на языке Promela приведен в листинге 1.1.

Листинг 1.1 — Пример программы на языке Promela

```
proctype check_triangle_type(int a, b, c) {
    // Проверка на существование треугольника
    if
    :: (a + b > c && a + c > b && b + c > a) -> {
        // Сортируем стороны, чтобы C всегда была самой большой
        if
        :: (a > b && a > c) -> {
            int temp = a;
            a = c;
            c = temp;
        }
        :: (b > a && b > c) -> {
            int temp = b;
            b = c;
            c = temp;
        }
        :: else -> skip;
        fi;

        // Проверка типа треугольника
        if
        :: (a * a + b * b == c * c) -> printf("Прямоугольный треугольник\n");
        :: (a * a + b * b > c * c) -> printf("Остроугольный треугольник\n");
        :: else -> printf("Тупоугольный треугольник\n");
        fi;
    }
    :: else -> printf("Это не треугольник\n");
    fi;
}

init {
    // Пример вызова программы с разными сторонами
    run check_triangle_type(3, 4, 5);  // Прямоугольный треугольник
    run check_triangle_type(5, 5, 8);  // Тупоугольный треугольник
    run check_triangle_type(3, 3, 4);  // Остроугольный треугольник
    run check_triangle_type(1, 2, 3);  // Это не треугольник
}
```

1.3 Описание модели

Данная модель описывает процесс, который определяет тип треугольника на основе длин его сторон.

Модель включает следующую функциональность:

— Процесс `check_triangle_type` принимает три целочисленных параметра (`a`, `b` и `c`), представляющих стороны треугольника.

— Сначала проверяется, могут ли данные стороны образовать допустимый треугольник, используя теорему о неравенстве треугольника.

— Если стороны образуют допустимый треугольник, они сортируются так, чтобы `c` всегда была самой длинной стороной.

— Затем проверяется тип треугольника:

  – Прямоугольный треугольник: если $a^2 + b^2 == c^2$

  – Остроугольный треугольник: если $a^2 + b^2 > c^2$

  – Тупоугольный треугольник: в остальных случаях

— Если стороны не образуют допустимый треугольник, выводится сообщение об этом.

— Процесс `init` демонстрирует использование `check_triangle_type` с различными наборами длин сторон:

  – (3, 4, 5): Прямоугольный треугольник

  – (5, 5, 8): Тупоугольный треугольник

  – (3, 3, 4): Остроугольный треугольник

  – (1, 2, 3): Не треугольник

1.4 Перечисление множества состояний

Множество состояний для процесса `check_triangle_type` включает следующие состояния:

— **S0**: Начальное состояние, где начинается процесс `check_triangle_type`.

— **S1**: Проверка условия существования треугольника (`a + b > c && a + c > b && b + c > a`).

— **S2**: Стороны не образуют треугольник.

— **S3**: Стороны образуют треугольник, сортировка сторон.

  – **S31**: Проверка, является ли сторона `a` самой длинной. Если да, меняем `a` и `c`.

  – **S32**: Проверка, является ли сторона `b` самой длинной. Если да, меняем `b` и `c`.

  – **S33**: Сторона `c` уже является самой длинной, сортировка не требуется.

— **S4***: Проверка типа треугольника ($a^2 + b^2$ по отношению к c^2).

— **S5***: Прямоугольный треугольник ($a^2 + b^2 == c^2$).

— **S6***: Остроугольный треугольник ($a^2 + b^2 > c^2$).

— **S7***: Тупоугольный треугольник ($a^2 + b^2 < c^2$).

— **S8****: Завершение процесса.

Переходы между состояниями происходят по следующим причинам:

— **S0** → **S1**: Начало процесса проверки треуголника по трем сторонам.

— **S1** → **S2**: Условие существования не выполнено, стороны не образуют треугольник.

— **S1** → **S3**: Условие существования выполнено, стороны образуют треугольник.

— **S3** → **S31**: Проверка, является ли сторона `a` самой длинной.

— **S3** → **S32**: Проверка, является ли сторона `b` самой длинной.

— **S3** → **S33**: Сторона `c` уже является самой длинной.

— **S31** → **S41**: Сортировка сторон завершена (меняем `a` и `c`).

— **S32** → **S42**: Сортировка сторон завершена (меняем `b` и `c`).

— **S33** → **S43**: Сортировка сторон завершена (сортировка не требуется).

— **S4x** → **S5x**: Условие прямоугольного треугольника выполнено.

— **S4x** → **S6x**: Условие остроугольного треугольника выполнено.

— **S4x** → **S7x**: Условие тупоугольного треугольника выполнено.

— **S5x** → **S8x1**: Печать результата «Прямоугольный треугольник».

— **S6x** → **S8x2**: Печать результата «Остроугольный треугольник».

— **S7x** → **S8x3**: Печать результата «Тупоугольный треугольник».

— **S2** → **S8**: Печать результата «Это не треугольник».

1.5 Граф переходов между состояниями модели

Граф переходов между состояниями модели приведен на рисунке 1.1.

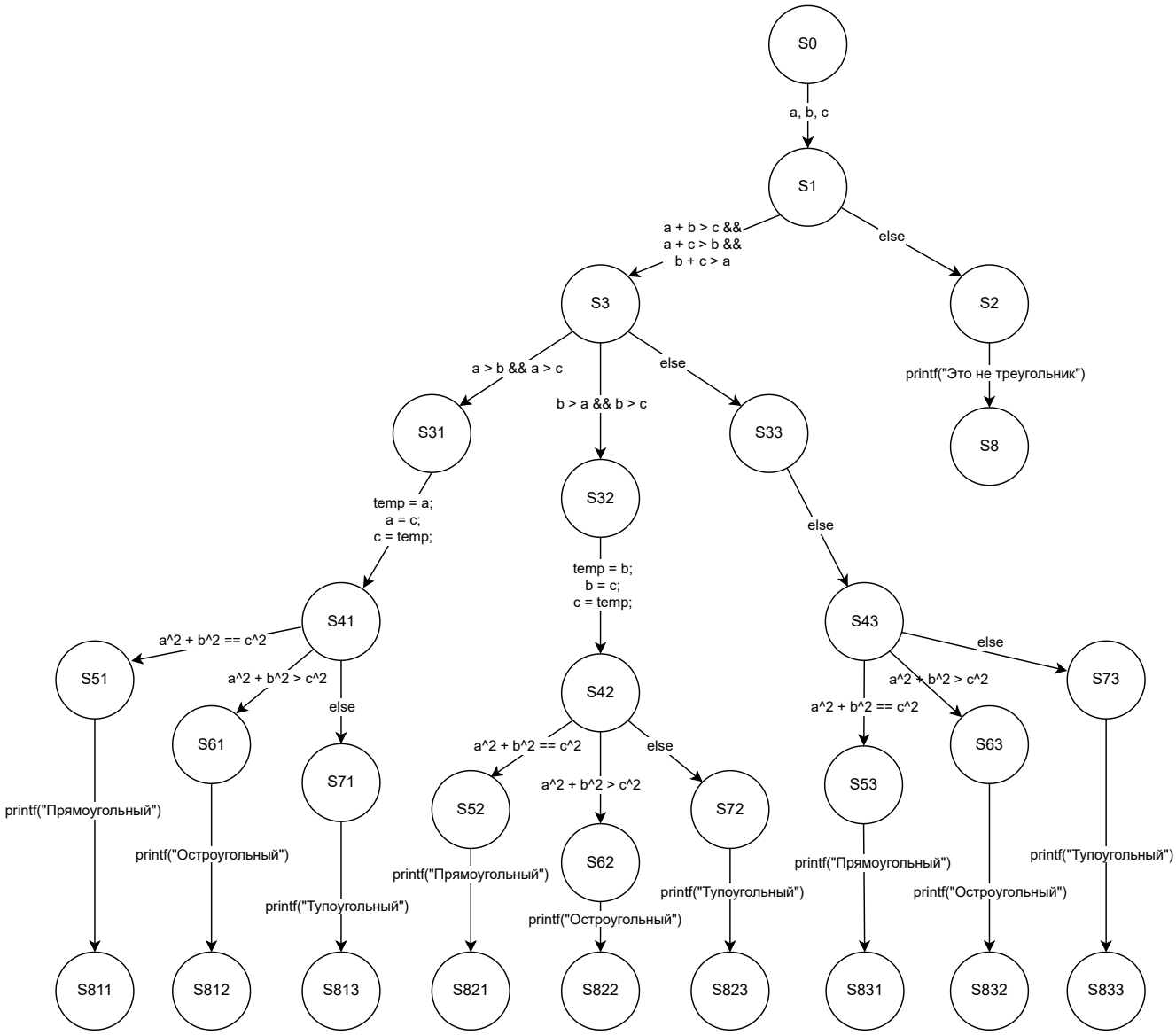

Рисунок 1.1 – Граф переходов между состояниями модели

1.6 Вывод

В ходе выполнения лабораторной работы были получены навыки работы с языком Promela и инструментом SPIN, а также выполнены следующие задачи:

— Разработан фрагмент кода на языке Promela, который демонстрирует проверку типа треугольника на основе длин его сторон.

— Описана модель, включающая функциональность процесса `check_triangle_type` и его взаимодействие с процессом `init`.

— Перечислено множество состояний процесса `check_triangle_type` с текстовым пояснением причин переходов между состояниями.

— Построен граф переходов между состояниями модели, иллюстрирующий возможные пути выполнения процесса.