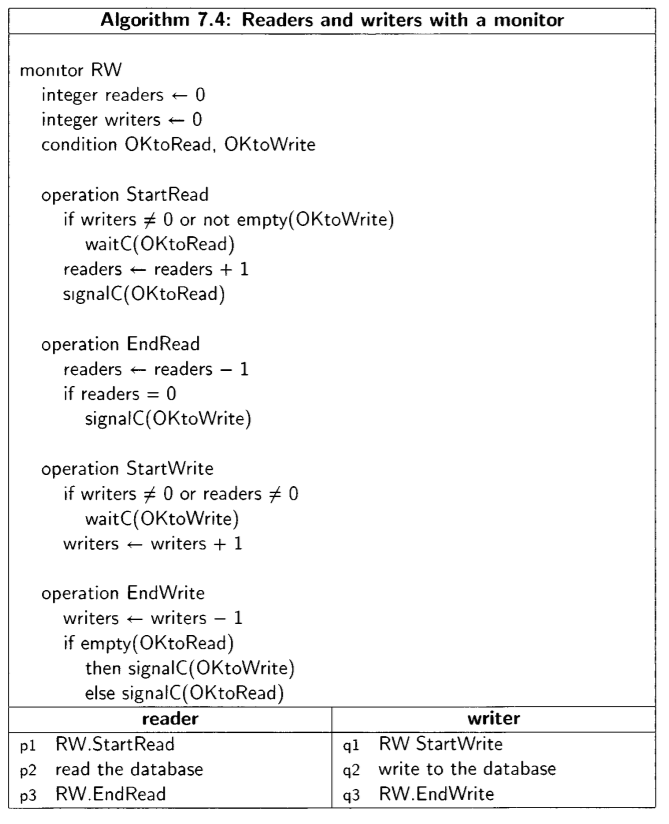
**Введение**

В данной курсовой работе рассматривается задача Readers and writers with a monitor, реализованная на языке Promela. Данная задача имеет много модификаций и в своей курсовой работе я решу одну из них.

**Постановка задачи**

Рассмотрим задачу 7.4 Readers and writers with a monitor, взятую из книги Ben-Ari. Есть несколько потоков, которые пытаются получить доступ к общему ресурсу. Другими словами проблема этой задачи аналогична проблеме взаимного исключения тем, что несколько процессов конкурируют за доступ к критической секции.

Однако в этой задаче мы делим процессы на два класса:

1) Процессы читатели, которые исключают писателей, но не других читателей

2) Процессы писатели, которые исключает как читателей, так и других писателей

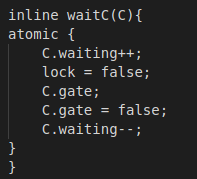
Необходимо обеспечить работу этих двух потоков с условием отсутствия приоритета.

**Решение**

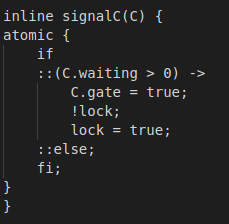
За основу возьмем стандартный алгоритм 7.4, взятый из книгиBen-Ari. Более подробно с языком Promela можно ознакомиться в пособии И.В. Шошминой и Ю.Г. Карпова «Введение в язык Promela и систему комплексной верификации Spin».

Алгоритм включает в себя протоколы(функции) Start Read/Write и End Read/Write. Используются мониторы со следующими параметрами: int readers, int writers (количество читателей и писателей), Conditonal OktoRead, Conditional OktoWrite (условные переменные).

В функции waitC(С) мы блокируем процесс, добавляя его в очередь(канал) командой C.waiting++, совобождаем монитор и затем ждем разблокировки процесса.

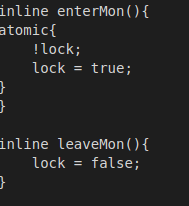


В функции signalC(С) мы проверяем пуст ли канал. Если канал не пустой, то разблокируем процесс и затем ждем возможность зайти в монитор.



В программе также создаем функции Start Read/Write и End Read/Write. Каждая из этих функции начинается с попадания в монитор, благодаря функции enterMon(), и заканчивается выходом из монитора, функцией leaveMon().

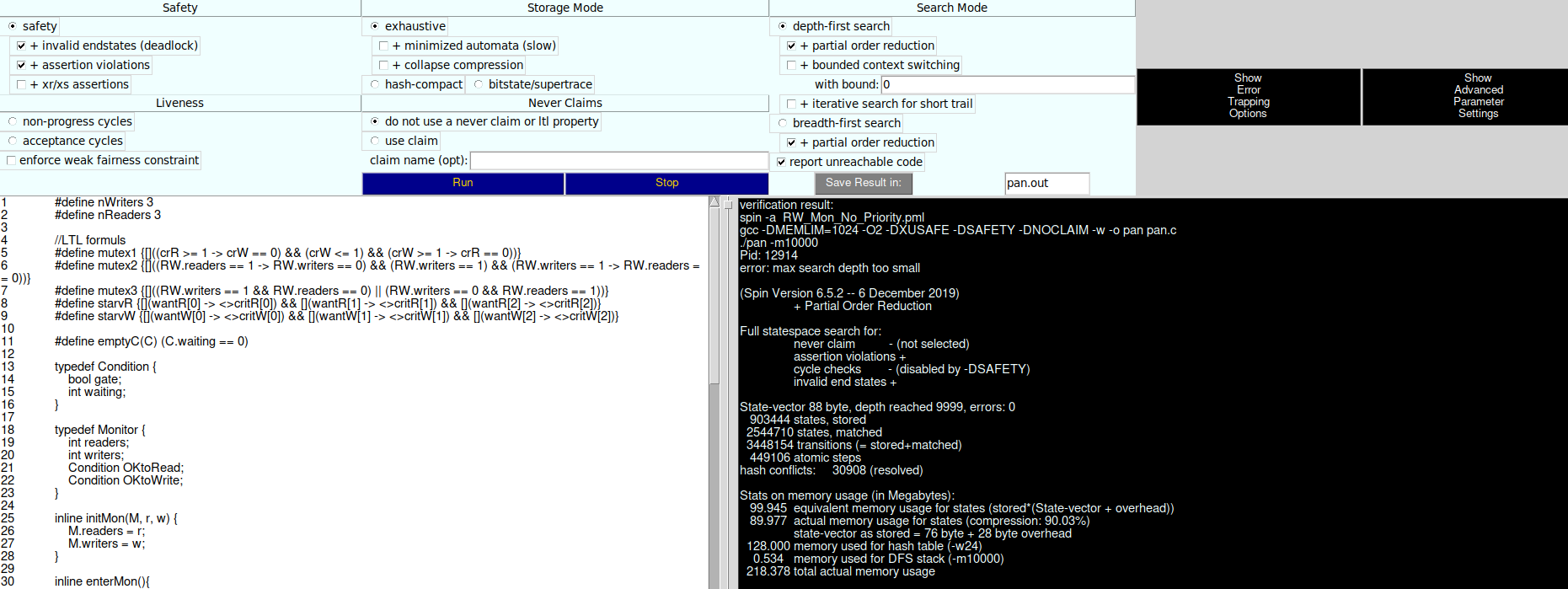
Вызов функции enterMon() позволяет процессу убедиться, что он может походить, сразу же снять это право у других процессов, благодаря флагу блокировки lock. Благодаря этой функции мы уверены, что никакой другой процесс не будет сейчас делать ход. После того, как функция завершилась, флаг lock снимается, что позволяет другим процессам запускать свои функции.



Когда писатель закончил писать, то он разблокирует читателя. Читатели во время исполнения функции signalC каскадно передают управление другим заблокированным читателям, чтобы прочитать изменившуюся информацию. Вход разделяет читателей на тех, кто был заблокирован в момент начала каскадной разблокировки и тех, кто был заблокирован позже, для того, чтобы у читателей не было приоритета. Condition в мониторах обычно является очередью, где процессы разблокируются в порядке поступления, но в данной задаче Condition реализован как счетчик и разблокирует тот процесс, который первым пытается прийти.

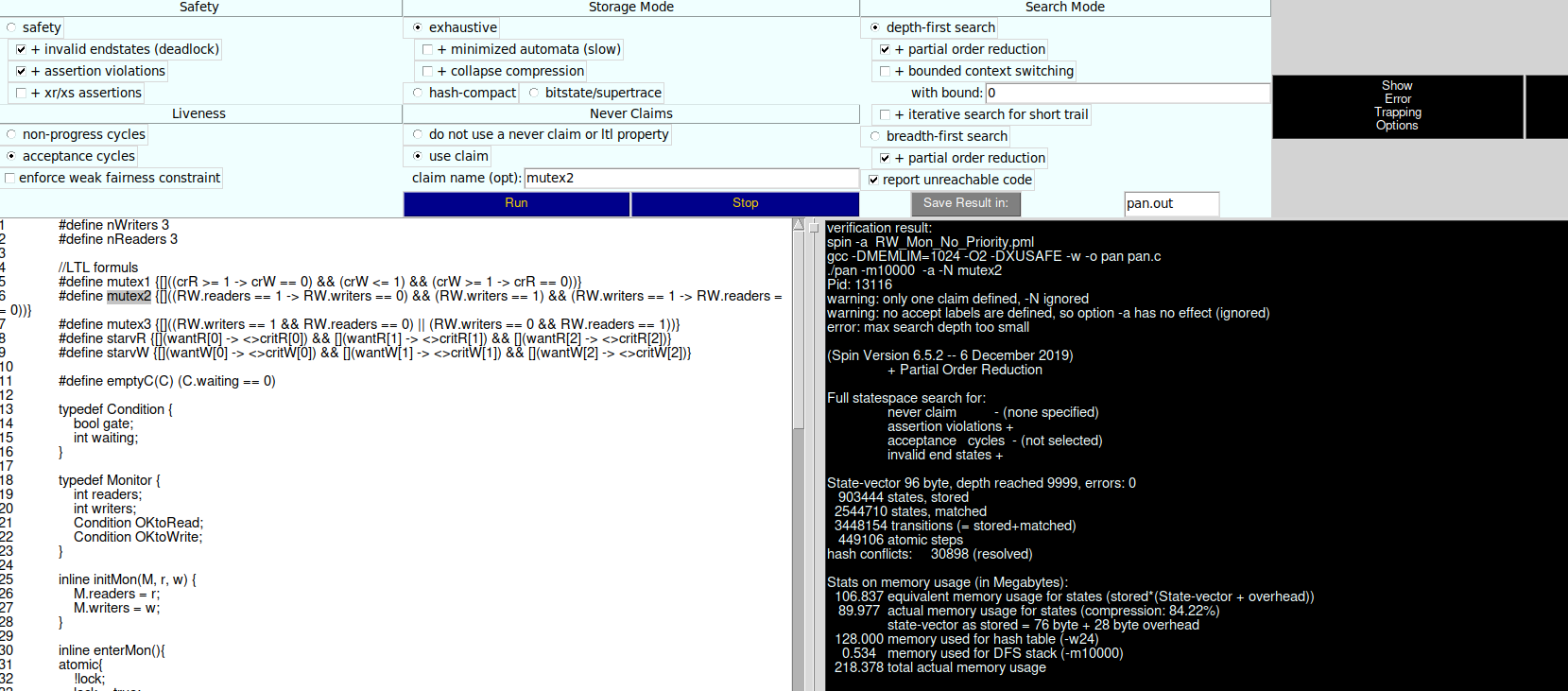
**Верификация**

1) Проверка свободы от взаимной блокировки



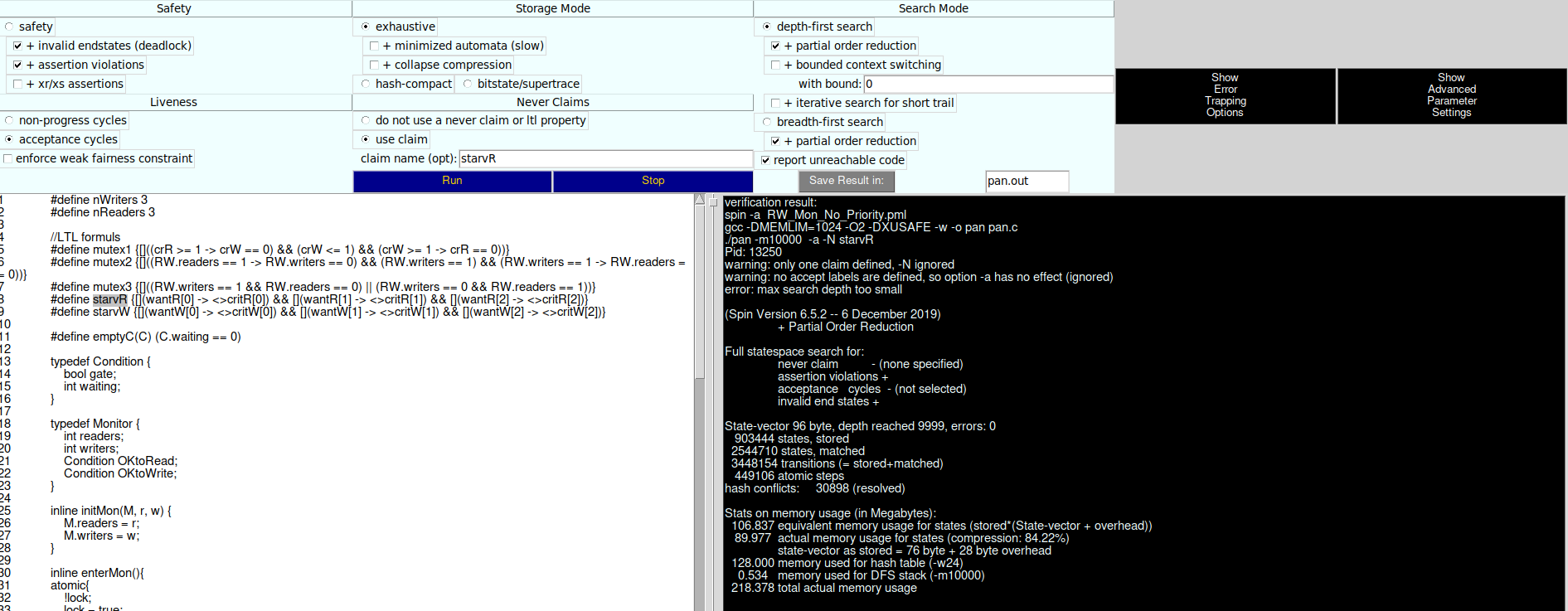
2) Проверка на взаимоисключающий доступ

#define mutex2 {[]((RW.readers == 1 -> RW.writers == 0) && (RW.writers == 1) && (RW.writers == 1 -> RW.readers == 0))}



3) Проверка свободы от голодания для читателей

#define starvR {[](wantR[0] -> <>critR[0]) && [](wantR[1] -> <>critR[1]) && [](wantR[2] -> <>critR[2])}



4) Проверка свободы от голодания для писателей

#define starvW {[](wantW[0] -> <>critW[0]) && [](wantW[1] -> <>critW[1]) && [](wantW[2] -> <>critW[2])}

