МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по индивидуальному домашнему заданию
по дисциплине «Верификация распределенных алгоритмов»
Тема: Разработка контроллера светофоров и его верификация
Вариант: 4

Студент гр. 9303 _____ Игнашов В.М.

Преподаватель _____ Шошмина И.В.

Санкт-Петербург
2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью работы является разработка модели контроллера перекрестка, регулируемого светофором, на языке Promela. Модель должна обрабатывать потоки машин по нескольким пересекающимся направлениям (в случае если направления движений не пересекаются – допустимо одновременное выполнение потоков). Предполагается, что потоки машин недетерминированные, процессы модели обрабатывают их параллельно для исключения последовательной обработки потоков. Необходимо реализовать модель таким образом, чтобы она соответствовала проверяемым требованиям: безопасность движения, живость движения и справедливость движения.

1. ПОСТРОЕНИЕ МОДЕЛИ

1.1. Описание задачи

Вариант: 4.

Четыре пересечения: NS/ED, SD/WN, SD/DN, WN/DE

Схема сложного перекрестка с указанными выше пересечениями представлена на рисунке 1. Синими точками отмечены конфликтующие направления из условия, красными – конфликтующие направления вне условия задачи.

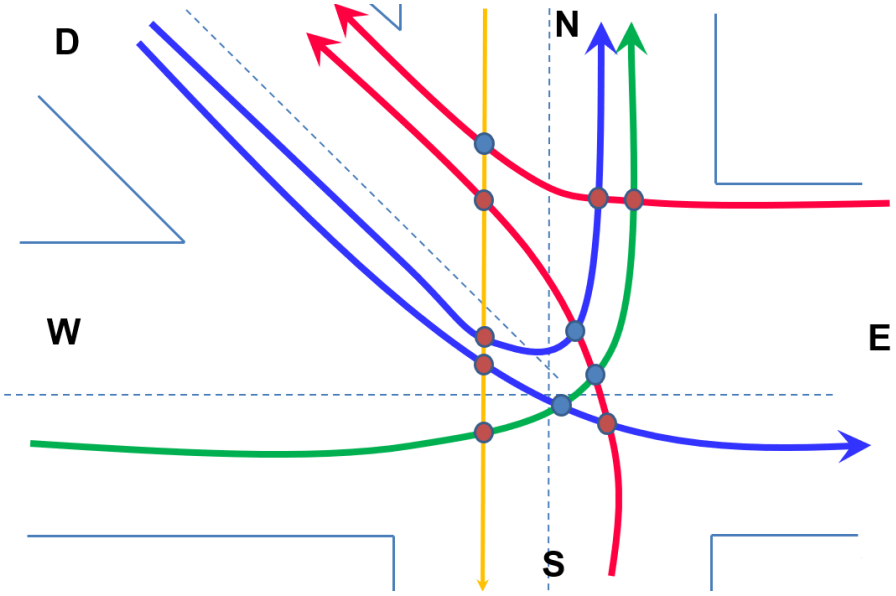

Рисунок 1. Схема перекрестка

Информация о том, какие направления должны быть закрыты при потоке машин для каждого направления, представлена в таблице 1.

Таблица 1. Конфликтующие направления

	NS	WN	SD	ED	DE	DN
NS		+	+	+	+	+
WN	+		+	+	+	
SD	+	+			+	+
ED	+	+				+
DE	+	+	+			
DN	+		+	+		

Дополнительные условия, которые необходимо учесть:

- Машины на каждом направлении движутся независимо
- Появление машины в каждом направлении регистрируется своим независимым датчиком движения

Исходный код реализованной модели представлен в приложении А.

1.2. Описание структур

Состояние каждого контроллера направления описывается реализованной структурой Direction, имеющей два булевых и одно числовое поле

```
typedef Direction {
	bool cars = 0;
	bool open = 0;
	byte conflicts = 0;
};
```

- cars – маркер, обозначающий наличие машин на этом направлении. Параметр использует логический тип данных в связи с отсутствием необходимости контролировать количество машин в потоке – независимо от их количества они все выполнят движение в случае открытого направления;
- open – маркер, обозначающий сигнал светофора (0 при закрытой дороге, 1 при открытой);
- conflicts – количество открытых конфликтующих направлений

Для синхронизации работы процессов используется структура QueueController, реализующая кольцевую очередь (FIFO). Использует поля с массивом для очереди и указателями на начало и конец очереди.

```
typedef QueueController{
	byte queue[PROCESSES_NUM];
	byte tail = 0;
	byte head = 0;
}
```

Также для структуры контроллера реализованы две функции pop_queue и push_queue для удаления и добавления нового элемента в очередь

соответственно. Выполнение обоих операций происходит атомарно для обеспечения синхронизации.

```
inline pop_queue(){
	atomic{
		controller.queue[controller.head] = 0;
		if
		:: controller.head == PROCESSES_NUM-1 -> controller.head = 0;
		:: else -> controller.head++;
		fi
	}
}
inline push_queue(id){
	atomic{
		controller.queue[controller.tail] = id
		if
		:: controller.tail == PROCESSES_NUM-1 -> controller.tail = 0;
		:: else -> controller.tail++;
		fi
	}
}
```

1.3. Описание процессов

В решении используются различные процессы, в совокупности выполняющие функционал светофора, обеспечивающие создание движения и управляющие потоками.

1.3.1. Процесс AddCarsToDirection

Данный процесс обеспечивает создание движения путем добавления направления в очередь ожидания и установления параметра cars в состояниях в значение 1 при отсутствии машин в данном направлении.

```
proctype AddCarsToDirection() {
	do
	:: !NS.cars -> {push_queue(1); NS.cars = 1;}
	:: !WN.cars -> {push_queue(2); WN.cars = 1;}
	:: !SD.cars -> {push_queue(3); SD.cars = 1;}
	:: !ED.cars -> {push_queue(4); ED.cars = 1;}
	:: !DE.cars -> {push_queue(5); DE.cars = 1;}
	:: !DN.cars -> {push_queue(6); DN.cars = 1;}
	od
}
```

1.3.2. Процесс Direction(DIR)

Были реализованы 6 схожих процессов для каждого из направлений (NS, WN, SD, ED, DE и DN соответственно), управляющие их состоянием.

Процесс выполняет действия в бесконечном цикле. В случае, если на направлении появляются машины – процесс выполняет следующую последовательность действий:

1. ожидает, пока станет первым в очереди направлением без конфликтов с другими направлениями;
2. увеличивает conflicts у каждого из конфликтующих направлений;
3. выходит из очереди;
4. открывает направление;
5. пропускает машины;
6. закрывает направление;
7. уменьшает conflicts у каждого из конфликтующих направлений.

Пример реализации процесса для направления NS представлен ниже.

```
proctype DirectionNS(){
	do
	:: NS.cars -> {
		if
		::controller.queue[controller.head] == 1 && !NS.conflicts -> {
			WN.conflicts++; SD.conflicts++; ED.conflicts++;
DE.conflicts++; DN.conflicts++;
			pop_queue();
			printf("NS: Opened")
			NS.open = 1;
			printf("NS: Released cars")
			NS.cars = 0;
			printf("NS: Closed")
			NS.open = 0;
			WN.conflicts--; SD.conflicts--; ED.conflicts--; DE.conflicts--
; DN.conflicts--;
		}
		fi
	}
	od
}
```

1.3.3. Процесс init

Процесс init отвечает за одновременное (atomic) создание всех процессов, включая AddCarsToDirection, и процессов, отвечающих за направления.

```
init {
	atomic {
		run DirectionNS();
		run DirectionWN();
		run DirectionSD();
		run DirectionED();
		run DirectionDE();
		run DirectionDN();

		run AddCarsToDirection();
	}
}
```

2. ВЕРИФИКАЦИЯ МОДЕЛИ

Для верификации реализованной модели было проверено, что для каждого из направлений NS, WN, SD, ED, DE и DN соблюдаются критерии безопасности, живости и справедливости. Критерии были описаны соответствующими LTL-формулами.

2.1. Свойство безопасности

Формулировка – «Никогда не случится ситуации, что направление открыто для движения и при этом какое-либо из конфликтующих направлений также открыто для движения»

LTL формулы для направлений представлены в таблице 2.

Таблица 2. Свойство безопасности

	LTL-формула
NS	$G(\overline{NS.open \;\&\&\; (WN.open \mid SD.open \mid ED.open \mid\; DE.open \mid DN.open)})$
WN	$G(\overline{WN.open \;\&\&\; (NS.open \mid SD.open \mid ED.open \mid DE.open\;)})$
SD	$G(\overline{SD.open \;\&\&\; (NS.open \mid WN.open \mid\; DE.open \mid DN.open)})$
ED	$G(\overline{ED.open \;\&\&\; (NS.open \mid WN.open \mid DN.open)})$
DE	$G(\overline{DE.open \;\&\&\; (NS.open \mid WN.open \mid SD.open)})$
DN	$G(\overline{DN.open \;\&\&\; (NS.open \mid SD.open \mid ED.open)})$

Описание LTL-формул в Promela:

```
ltl safety0 { [] ! (NS.open && (WN.open || SD.open || ED.open || DE.open ||
DN.open))}
ltl safety1 { [] ! (WN.open && (NS.open || SD.open || ED.open || DE.open))}
ltl safety2 { [] ! (SD.open && (NS.open || WN.open || DE.open || DN.open))}
ltl safety3 { [] ! (ED.open && (NS.open || WN.open || DN.open))}
ltl safety4 { [] ! (DE.open && (NS.open || WN.open || SD.open))}
ltl safety5 { [] ! (DN.open && (NS.open || SD.open || ED.open))}
```

2.2. Свойство живости

Формулировка – «Всегда правда, что если в направлении присутствуют машины и направление закрыто – то когда-нибудь направление откроется»

LTL-формула одинакова для всех направлений (DIR - направление):

$$G((DIR.cars \;\&\&\; \overline{DIR.open}) \rightarrow F(DIR.open))$$

```
ltl liveness0 { [] ((NS.cars && !NS.open) -> <> (NS.open))}
ltl liveness1 { [] ((WN.cars && !WN.open) -> <> (WN.open))}
ltl liveness2 { [] ((SD.cars && !SD.open) -> <> (SD.open))}
ltl liveness3 { [] ((ED.cars && !ED.open) -> <> (ED.open))}
ltl liveness4 { [] ((DE.cars && !DE.open) -> <> (DE.open))}
ltl liveness5 { [] ((DN.cars && !DN.open) -> <> (DN.open))}
```

2.3. Свойство справедливости

Формулировка – «Всегда в будущем либо направление закрыто, либо на направлении нет машин»

LTL-формула одинакова для всех направлений (DIR - направление):

$$GF(\overline{DIR.open\ \&\&DIR.cars})$$

```
ltl fairness0 { [] (<> (! (NS.open && NS.cars)))}
ltl fairness1 { [] (<> (! (WN.open && WN.cars)))}
ltl fairness2 { [] (<> (! (SD.open && SD.cars)))}
ltl fairness3 { [] (<> (! (ED.open && ED.cars)))}
ltl fairness4 { [] (<> (! (DE.open && DE.cars)))}
ltl fairness5 { [] (<> (! (DN.open && DN.cars)))}
```

2.4. Верификация в Spin

При верификации модели с помощью Spin были установлены параметры, представленные на рисунке 2.

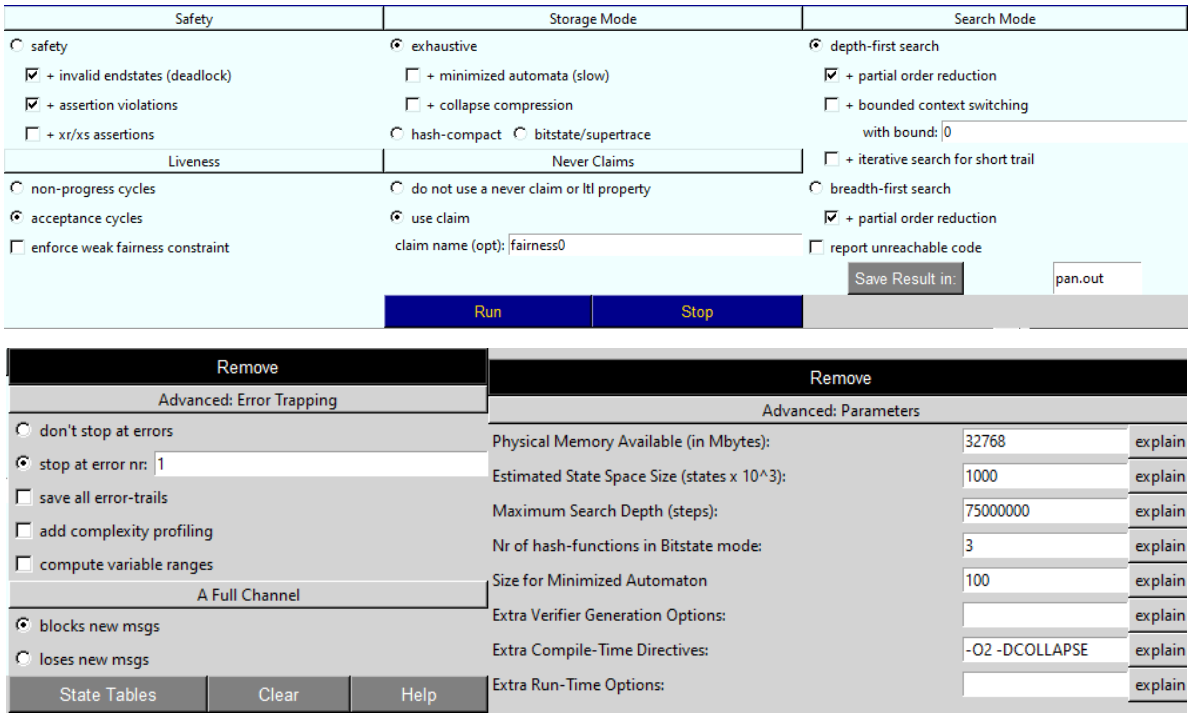

Рисунок 2. Параметры для верификации свойств

Функция «report unreachable code» отключена по причине, что предполагается, что модель работает бесконечно, а значит процессы не приходят в состояние “-end-“.

По причине того, что для верификации модели необходимо использовать большие вычислительные ресурсы, параметры используемой памяти и максимальной глубины поиска увеличены, а также использован аргумент DCOLLAPSE при компиляции для компрессии.

Результаты верификации на примере направления NS свойств безопасности (safety0), безопасности (liveness0) и справедливости (fairness0) представлены в приложениях B, C и D соответственно.

ЗАКЛЮЧЕНИЕ

В результате выполнения данной работы была выполнена задача моделирования сложного перекрестка, содержащего конфликтующие направления. По итогам ее выполнения была реализована модель перекрестка на языке Promela, использующая 1 процесса, управляющий внешней средой и 6 процессов, управляющих направлениями, заданными в условии задачи.

Для итоговой модели была проведена верификация свойств безопасности, живости и справедливости, выраженных в LTL-формулах. По результатам верификации можно сказать, что модель корректна.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Карпов Ю.Г., Шошмина И.В. Верификация распределенных систем: учеб. пособие СПб.: Изд-во Политехнического университета, 2011. 212 с.

2. Шошмина И.В., Карпов Ю.Г. Введение в язык Promela и систему комплексной верификации Spin: учеб. пособие СПб.: Изд-во Политехнического университета, 2009

3. Карпов Ю.Г. Model checking. Верификация параллельных и распределенных программных систем. // БХВ-Петербург, 2009, 520 с.

4. Документация к Promela // Promela Manual Pages [Электронный ресурс]. URL: https://spinroot.com/spin/Man/promela.html (дата обращения: 26.04.2024)

5. Документация к Spin // Spin Online References [Электронный ресурс]. URL: https://spinroot.com/spin/Man/ (дата обращения: 26.04.2024)

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД МОДЕЛИ

Название файла - traffic-light.pml

```
#define PROCESSES_NUM 6
// NS WN SD ED DE DN
typedef Direction {
	bool cars = 0;
	bool open = 0;
	byte conflicts = 0;
};
typedef QueueController{
	byte queue[PROCESSES_NUM];
	byte tail = 0;
	byte head = 0;
}
Direction NS, WN, SD, ED, DE, DN;
QueueController controller;
inline pop_queue(){
	atomic{
		controller.queue[controller.head] = 0;
		if
		:: controller.head == PROCESSES_NUM-1 -> controller.head = 0;
		:: else -> controller.head++;
		fi
	}
}
inline push_queue(id){
	atomic{
		controller.queue[controller.tail] = id
		if
		:: controller.tail == PROCESSES_NUM-1 -> controller.tail = 0;
		:: else -> controller.tail++;
		fi
	}
}
proctype AddCarsToDirection() {
	do
	:: !NS.cars -> {push_queue(1); NS.cars = 1;}
```

```
      :: !WN.cars -> {push_queue(2); WN.cars = 1;}
      :: !SD.cars -> {push_queue(3); SD.cars = 1;}
      :: !ED.cars -> {push_queue(4); ED.cars = 1;}
      :: !DE.cars -> {push_queue(5); DE.cars = 1;}
      :: !DN.cars -> {push_queue(6); DN.cars = 1;}
      od
}
proctype DirectionNS(){
      do
      :: NS.cars -> {
            if
            ::controller.queue[controller.head] == 1 && !NS.conflicts -> {
                  WN.conflicts++; SD.conflicts++; ED.conflicts++;
DE.conflicts++; DN.conflicts++;
                  pop_queue();
                  NS.open = 1; NS.cars = 0; NS.open = 0;
                  WN.conflicts--; SD.conflicts--; ED.conflicts--;
DE.conflicts--; DN.conflicts--;
            }
            fi
      }
      od
}
proctype DirectionWN(){
      do
      :: WN.cars -> {
            if
            ::controller.queue[controller.head] == 2 && !WN.conflicts -> {
                  NS.conflicts++; SD.conflicts++; ED.conflicts++;
DE.conflicts++;
                  pop_queue();
                  WN.open = 1; WN.cars = 0; WN.open = 0;
                  NS.conflicts--; SD.conflicts--; ED.conflicts--;
DE.conflicts--;
            }
            fi
      }
      od
```

```
}
proctype DirectionSD(){
	do
	:: SD.cars -> {
		if
		::controller.queue[controller.head] == 3 && !SD.conflicts -> {
			NS.conflicts++; WN.conflicts++; DE.conflicts++;
DN.conflicts++;
			pop_queue();
			SD.open = 1; SD.cars = 0; SD.open = 0;
			NS.conflicts--; WN.conflicts--; DE.conflicts--;
DN.conflicts--;
		}
		fi
	}
	od
}
proctype DirectionED(){
	do
	:: ED.cars -> {
		if
		::controller.queue[controller.head] == 4 && !ED.conflicts -> {
			NS.conflicts++; WN.conflicts++; DN.conflicts++;
			pop_queue();
			ED.open = 1; ED.cars = 0; ED.open = 0;
			NS.conflicts--; WN.conflicts--; DN.conflicts--;
		}
		fi
	}
	od
}
proctype DirectionDE(){
	do
	:: DE.cars -> {
		if
		::controller.queue[controller.head] == 5 && !DE.conflicts -> {
			NS.conflicts++; WN.conflicts++; SD.conflicts++;
			pop_queue();
```

```
                DE.open = 1; DE.cars = 0; DE.open = 0;
                NS.conflicts--; WN.conflicts--; SD.conflicts--;
            }
            fi
        }
        od
}
proctype DirectionDN(){
        do
        :: DN.cars -> {
            if
            ::controller.queue[controller.head] == 6 && !DN.conflicts -> {
                NS.conflicts++; SD.conflicts++; ED.conflicts++;
                pop_queue();
                DN.open = 1; DN.cars = 0; DN.open = 0;
                NS.conflicts--; SD.conflicts--; ED.conflicts--;
            }
            fi
        }
        od
}
init{
        atomic {
            run DirectionNS();
            run DirectionWN();
            run DirectionSD();
            run DirectionED();
            run DirectionDE();
            run DirectionDN();
            run AddCarsToDirection()
        }
}
// LTL-Checks
// Safety: Always (if direction is open then conflicting one is open) is
False
ltl safety0 { [] ! (NS.open && (WN.open || SD.open || ED.open || DE.open
|| DN.open))}
```

```
ltl safety1 { [] ! (WN.open && (NS.open || SD.open || ED.open ||
DE.open))}
ltl safety2 { [] ! (SD.open && (NS.open || WN.open || DE.open ||
DN.open))}
ltl safety3 { [] ! (ED.open && (NS.open || WN.open || DN.open))}
ltl safety4 { [] ! (DE.open && (NS.open || WN.open || SD.open))}
ltl safety5 { [] ! (DN.open && (NS.open || SD.open || ED.open))}
// Liveness: Always (if there are cars in direction and it is closed then
later it will open) is True
ltl liveness0 { [] ((NS.cars && !NS.open) -> <> (NS.open))}
ltl liveness1 { [] ((SD.cars && !WN.open) -> <> (WN.open))}
ltl liveness2 { [] ((SD.cars && !SD.open) -> <> (SD.open))}
ltl liveness3 { [] ((ED.cars && !ED.open) -> <> (ED.open))}
ltl liveness4 { [] ((DE.cars && !DE.open) -> <> (DE.open))}
ltl liveness5 { [] ((DN.cars && !DN.open) -> <> (DN.open))}
// Fairness: Always in the future either there won't be any cars either
the direction will close
ltl fairness0 { [] (<> (! (NS.open && NS.cars)))}
ltl fairness1 { [] (<> (! (WN.open && SD.cars)))}
ltl fairness2 { [] (<> (! (SD.open && SD.cars)))}
ltl fairness3 { [] (<> (! (ED.open && ED.cars)))}
ltl fairness4 { [] (<> (! (DE.open && DE.cars)))}
ltl fairness5 { [] (<> (! (DN.open && DN.cars)))}
```

ПРИЛОЖЕНИЕ В

ВЕРИФИКАЦИЯ СВОЙСТВА БЕЗОПАСНОСТИ (NS)

```
gcc -DMEMLIM=32768 -O2 -DCOLLAPSE -DXUSAFE -w -o pan pan.c
./pan -m75000000  -a -n -N safety0
Pid: 20328
pan: ltl formula safety0
Depth= 2010722 States=    1e+06 Transitions= 2.12e+06 Memory=  4198.764
     t=    0.762 R=   1e+06
                                        …
Depth= 69357057 States= 2.71e+08 Transitions= 1.32e+09 Memory= 20548.557
     t=      770 R=   4e+05

(Spin Version 6.5.1 -- 20 December 2019)
     + Partial Order Reduction
     + Compression

Full statespace search for:
     never claim          + (safety0)
     assertion violations  + (if within scope of claim)
     acceptance   cycles   + (fairness disabled)
     invalid end states    - (disabled by never claim)

State-vector 120 byte, depth reached 69357057, errors: 0
2.7149184e+08 states, stored
1.093715e+09 states, matched
1.3652068e+09 transitions (= stored+matched)
1.6308217e+08 atomic steps
hash conflicts: 1.4824998e+09 (resolved)

Stats on memory usage (in Megabytes):
38319.390  equivalent memory usage for states (stored*(State-vector +
overhead))
16063.677  actual memory usage for states (compression: 41.92%)
           state-vector as stored = 34 byte + 28 byte overhead
  512.000  memory used for hash table (-w26)
 4005.432  memory used for DFS stack (-m75000000)
    4.329  memory lost to fragmentation
20576.780  total actual memory usage


nr of templates: [ 0:globals 1:chans 2:procs ]
collapse counts: [ 0:15781507 2:13 3:16 4:14 5:14 6:12 7:12 8:12 9:2 10:1
]

pan: elapsed time 833 seconds
No errors found -- did you verify all claims?
```

ПРИЛОЖЕНИЕ С

ВЕРИФИКАЦИЯ СВОЙСТВА ЖИВОСТИ (NS)

```
gcc -DMEMLIM=32768 -O2 -DCOLLAPSE -DXUSAFE -w -o pan pan.c
./pan -m75000000  -a -n -c1 -N liveness0
Pid: 8048
pan: ltl formula liveness0
Depth=     299 States=    1e+06 Transitions= 4.51e+06 Memory=  4163.217
     t=     1.03 R=   1e+06
                                        …
Depth= 69357057 States= 6.66e+08 Transitions=  4.4e+09 Memory= 31089.475
     t= 2.25e+03 R=   3e+05

(Spin Version 6.5.1 -- 20 December 2019)
     + Partial Order Reduction
     + Compression

Full statespace search for:
     never claim          + (liveness0)
     assertion violations  + (if within scope of claim)
     acceptance   cycles   + (fairness disabled)
     invalid end states    - (disabled by never claim)

State-vector 120 byte, depth reached 69357057, errors: 0
4.6923096e+08 states, stored (6.6697e+08 visited)
3.7952671e+09 states, matched
4.4622371e+09 transitions (= visited+matched)
4.8926869e+08 atomic steps
hash conflicts: 6.5179122e+09 (resolved)

Stats on memory usage (in Megabytes):
66229.040  equivalent memory usage for states (stored*(State-vector +
overhead))
26629.706  actual memory usage for states (compression: 40.21%)
           state-vector as stored = 32 byte + 28 byte overhead
  512.000  memory used for hash table (-w26)
 4005.432  memory used for DFS stack (-m75000000)
    7.174  memory lost to fragmentation
31139.964  total actual memory usage


nr of templates: [ 0:globals 1:chans 2:procs ]
collapse counts: [ 0:15781507 2:13 3:16 4:14 5:14 6:12 7:12 8:12 9:2 16:2
]

pan: elapsed time 2.39e+03 seconds
No errors found -- did you verify all claims?
```

ПРИЛОЖЕНИЕ D

ВЕРИФИКАЦИЯ СВОЙСТВА СПРАВЕДЛИВОСТИ (NS)

```
gcc -DMEMLIM=32768 -O2 -DCOLLAPSE -DXUSAFE -w -o pan pan.c
./pan -m75000000  -a -n -c1 -N fairness0
Pid: 5688
pan: ltl formula fairness0
Depth= 1939430 States=    1e+06 Transitions= 2.11e+06 Memory=  4197.104
     t=    0.657 R=   2e+06
                                        …
Depth= 69357057 States= 3.26e+08 Transitions= 1.53e+09 Memory= 22013.108
     t=      856 R=   4e+05

(Spin Version 6.5.1 -- 20 December 2019)
     + Partial Order Reduction
     + Compression

Full statespace search for:
     never claim          + (fairness0)
     assertion violations  + (if within scope of claim)
     acceptance   cycles   + (fairness disabled)
     invalid end states    - (disabled by never claim)

State-vector 120 byte, depth reached 69357057, errors: 0
2.9908709e+08 states, stored (3.26659e+08 visited)
1.2613277e+09 states, matched
1.5879864e+09 transitions (= visited+matched)
1.8341664e+08 atomic steps
hash conflicts: 1.7379529e+09 (resolved)

Stats on memory usage (in Megabytes):
42214.288  equivalent memory usage for states (stored*(State-vector +
overhead))
17538.157  actual memory usage for states (compression: 41.55%)
           state-vector as stored = 33 byte + 28 byte overhead
  512.000  memory used for hash table (-w26)
 4005.432  memory used for DFS stack (-m75000000)
    4.688  memory lost to fragmentation
22050.901  total actual memory usage


nr of templates: [ 0:globals 1:chans 2:procs ]
collapse counts: [ 0:15781507 2:13 3:16 4:14 5:14 6:12 7:12 8:12 9:2 22:2
]

pan: elapsed time 922 seconds
No errors found -- did you verify all claims?
```