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.

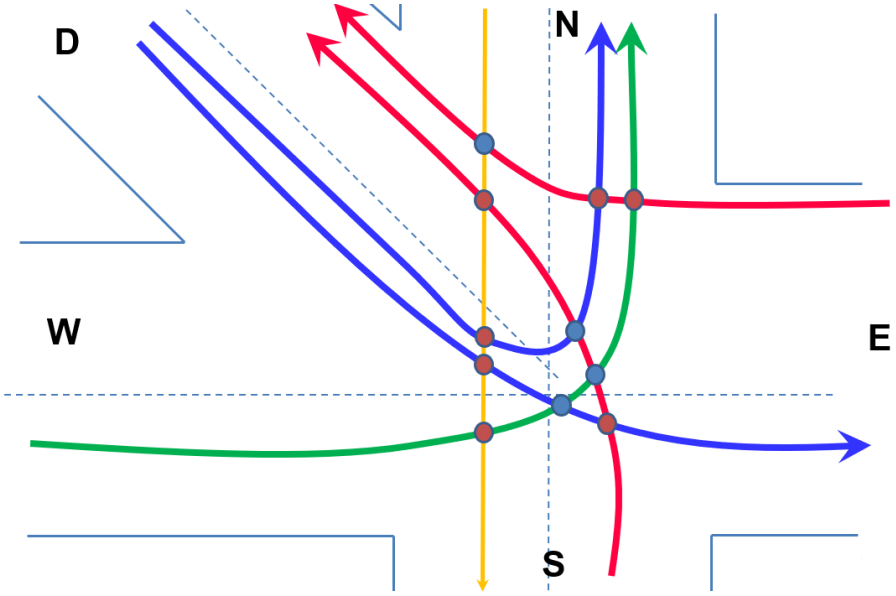

Рисунок 1. Схема перекрестка

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 три булевых поля

```
typedef Direction {
	bool cars = 0;
	bool open = 0;
	bool completed = 1;
};
```

- 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);
- completed – маркер, обозначающий завершение действия. Обновляется на 0 в случае, если процесс MarkAsNotCompleted определил, что процессы завершили свою работу на этой «итерации». Таким образом обеспечивается выполнение критерия справедливости и критерия живости.

Для синхронизации работы процессов используются семафоры. Его структура содержит только счетчик C. Реализованы две inline функции, позволяющие захватить и отпустить контроль.

С помощью таких семафоров выполняется блокировка конфликтующих направлений, для этого созданы переменные NS_WN, NS_SD, NS_ED, NS_DE, NS_DN, WN_SD, WN_ED, WN_DE, SD_DE, SD_DN и ED_DN для каждой пары.

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 установления параметра cars в состояниях в значение 1 при отсутствии машин в данном направлении.

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

1.3.2. Процесс MarkAsNotCompleted

Процесс обновляет значения completed у всех направлений в случае, если каждый из них на этой «итерации» завершил свою работу. Таким образом, между ситуациями, когда ни одно из направлений не должно быть открыто, каждое из направлений выполнит свою работу. Процесс использует специальный маркер COMPLETED_MARKER, определяющий количество процессов, завершивших работу.

```
proctype MarkAsNotCompleted() {
    do
    :: COMPLETED_MARKER == PROCESSES_NUM -> {
            COMPLETED_MARKER = 0;
           NS.completed = 0; WN.completed = 0; SD.completed = 0;
           ED.completed = 0; DE.completed = 0; DN.completed = 0;
       }
    od
}
```

1.3.3. Процессы Direction(DIR)

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

Процесс выполняет действия в бесконечном цикле в случае, если свою задачу не завершил (completed==0). Если в направлении присутствуют машины – процесс выполняет следующую последовательность действий:

1. ожидает, пока может взять контроль над пересечениями с конфликтными направлениями, последовательно берет контроль.
2. открывает направление;
3. пропускает поток машин;
4. закрывает направление.

Далее независимо от того, присутствовали машины или нет он завершает работу, изменяя соответствующие маркеры. Пример реализации процесса для направления NS представлен ниже.

```
proctype DirectionNS() {
	do
	:: (!NS.completed) -> {
		if
		:: NS.cars -> {
			acquire(NS_WN); acquire(NS_SD); acquire(NS_ED);
acquire(NS_DN); acquire(NS_DE);
			NS.open = 1;
			printf("Opened NS direction")
			NS.cars = 0;
			printf("Released cars for NS direction")
			NS.open = 0;
			printf("Closed NS direction")
			release(NS_WN); release(NS_SD); release(NS_ED);
release(NS_DN); release(NS_DE);
		}
		fi
		NS.completed = 1; COMPLETED_MARKER = COMPLETED_MARKER + 1;
		printf("Finished NS direction")
	}
	od
}
```

1.3.4. Процесс init

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

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

		run AddCarsToDirection();
		run MarkAsNotCompleted();
	}
}
```

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.

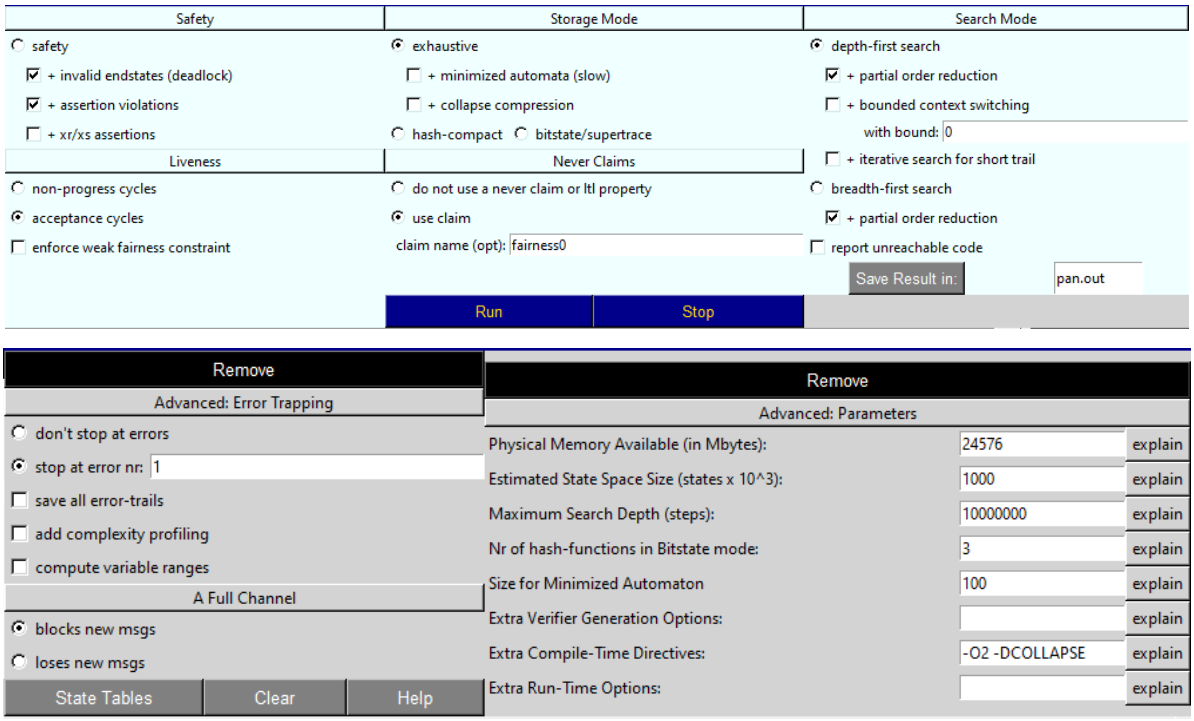

Рисунок 2. Параметры для верификации свойств

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 2 процесса, управляющих 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
typedef Direction {
	bool cars = 0;
	bool open = 0;
	bool completed = 1;
};
typedef Semaphore{
	byte C = 1;
}
inline acquire(sem){
	atomic{
		sem.C > 0;
		sem.C--;
	}
}
inline release(sem){
	sem.C++;
}
Semaphore NS_WN, NS_SD, NS_ED, NS_DE, NS_DN;
Semaphore WN_SD, WN_ED, WN_DE;
Semaphore SD_DE, SD_DN;
Semaphore ED_DN;
byte COMPLETED_MARKER = PROCESSES_NUM
Direction NS, WN, SD, ED, DE, DN;
// 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 { [] ((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))}
// 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 && 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)))}
// Process that adds new cars to any empty direction
proctype AddCarsToDirection() {
      do
      :: !NS.cars -> {NS.cars = 1;}
      :: !WN.cars -> {WN.cars = 1;}
      :: !SD.cars -> {SD.cars = 1;}
      :: !ED.cars -> {ED.cars = 1;}
      :: !DE.cars -> {DE.cars = 1;}
      :: !DN.cars -> {DN.cars = 1;}
      od
}
// Process that marks all the directions as not completed
proctype MarkAsNotCompleted() {
    do
    :: COMPLETED_MARKER == PROCESSES_NUM -> {
           COMPLETED_MARKER = 0;
           NS.completed = 0; WN.completed = 0; SD.completed = 0;
           ED.completed = 0; DE.completed = 0; DN.completed = 0;
       }
    od
}
```

```
proctype DirectionNS() {
      do
      :: (!NS.completed) -> {
            if
            :: NS.cars -> {
                  acquire(NS_WN); acquire(NS_SD); acquire(NS_ED);
acquire(NS_DN); acquire(NS_DE);
                  NS.open = 1;
                  printf("Opened NS direction")
                  NS.cars = 0;
                  printf("Released cars for NS direction")
                  NS.open = 0;
                  printf("Closed NS direction")
                  release(NS_WN); release(NS_SD); release(NS_ED);
release(NS_DN); release(NS_DE);
            }
            fi
            NS.completed = 1; COMPLETED_MARKER = COMPLETED_MARKER + 1;
            printf("Finished NS direction")
      }
      od
}
proctype DirectionWN() {
      do
      :: (!WN.completed) -> {
            if
            :: WN.cars -> {
                  acquire(NS_WN); acquire(WN_DE); acquire(WN_ED);
acquire(WN_SD);
                  WN.open = 1;
                  printf("Opened WN direction")
                  WN.cars = 0;
                  printf("Released cars for WN direction")
                  WN.open = 0;
                  printf("Closed WN direction")
                  release(NS_WN); release(WN_DE); release(WN_ED);
release(WN_SD);
            }
```

```
            fi
            WN.completed = 1; COMPLETED_MARKER = COMPLETED_MARKER + 1;
            printf("Finished WN direction")
      }
      od
}
proctype DirectionSD() {
      do
      :: (!SD.completed) -> {
            if
            :: SD.cars -> {
                  acquire(NS_SD); acquire(WN_SD); acquire(SD_DE);
acquire(SD_DN);
                  SD.open = 1;
                  printf("Opened SD direction")
                  SD.cars = 0;
                  printf("Released cars for SD direction")
                  SD.open = 0;
                  printf("Closed SD direction")
                  release(NS_SD); release(WN_SD); release(SD_DE);
release(SD_DN);
            }
            fi
            SD.completed = 1; COMPLETED_MARKER = COMPLETED_MARKER + 1;
            printf("Finished SD direction")
      }
      od
}
proctype DirectionED() {
      do
      :: (!ED.completed) -> {
            if
            :: ED.cars -> {
                  acquire(NS_ED); acquire(WN_ED); acquire(ED_DN);
                  ED.open = 1;
                  printf("Opened ED direction")
                  ED.cars = 0;
                  printf("Released cars for ED direction")
```

```
                ED.open = 0;
                printf("Closed ED direction")
                release(NS_ED); release(WN_ED); release(ED_DN);
            }
            fi
            ED.completed = 1; COMPLETED_MARKER = COMPLETED_MARKER + 1;
            printf("Finished ED direction")
        }
        od
}
proctype DirectionDE() {
        do
        :: (!DE.completed) -> {
            if
            :: DE.cars -> {
                acquire(NS_DE); acquire(WN_DE); acquire(SD_DE);
                DE.open = 1;
                printf("Opened DE direction")
                DE.cars = 0;
                printf("Released cars for DE direction")
                DE.open = 0;
                printf("Closed DE direction")
                release(NS_DE); release(WN_DE); release(SD_DE);
            }
            fi
            DE.completed = 1; COMPLETED_MARKER = COMPLETED_MARKER + 1;
            printf("Finished DE direction")
        }
        od
}
proctype DirectionDN() {
        do
        :: (!DN.completed) -> {
            if
            :: DN.cars -> {
                acquire(NS_DN); acquire(SD_DN); acquire(ED_DN);
                DN.open = 1;
                printf("Opened DN direction")
```

```
                DN.cars = 0;
                printf("Released cars for DN direction")
                DN.open = 0;
                printf("Closed DN direction")
                release(NS_DN); release(SD_DN); release(ED_DN);
            }
            fi
            DN.completed = 1; COMPLETED_MARKER = COMPLETED_MARKER + 1;
            printf("Finished DN direction")
        }
        od
}
init {
        atomic {
            run DirectionNS();
            run DirectionWN();
            run DirectionSD();
            run DirectionED();
            run DirectionDE();
            run DirectionDN();

            run AddCarsToDirection();
            run MarkAsNotCompleted();
        }
}
```

ПРИЛОЖЕНИЕ В

ВЕРИФИКАЦИЯ СВОЙСТВА БЕЗОПАСНОСТИ (NS)

```
gcc -DMEMLIM=16384 -O2 -DCOLLAPSE -DXUSAFE -w -o pan pan.c
./pan -m10000000  -a -n -N safety0
Pid: 8332
pan: ltl formula safety0
Depth=   12748 States=    1e+06 Transitions= 4.88e+06 Memory=   723.972
     t=     1.33 R=   8e+05
                                        …
Depth=   54296 States= 2.51e+08 Transitions= 1.53e+09 Memory= 14749.644
     t=      834 R=   3e+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 104 byte, depth reached 54296, errors: 0
2.5168749e+08 states, stored
1.2859424e+09 states, matched
1.5376299e+09 transitions (= stored+matched)
        7 atomic steps
hash conflicts: 1.6552311e+09 (resolved)

Stats on memory usage (in Megabytes):
31683.682  equivalent memory usage for states (stored*(State-vector +
overhead))
13744.472  actual memory usage for states (compression: 43.38%)
           state-vector as stored = 29 byte + 28 byte overhead
  512.000  memory used for hash table (-w26)
  534.058  memory used for DFS stack (-m10000000)
    4.167  memory lost to fragmentation
14786.362  total actual memory usage


nr of templates: [ 0:globals 1:chans 2:procs ]
collapse counts: [ 0:2658271 2:7 3:8 4:17 5:15 6:15 7:13 8:13 9:13 10:2
11:1 ]

pan: elapsed time 838 seconds
No errors found -- did you verify all claims?
```

ПРИЛОЖЕНИЕ С

ВЕРИФИКАЦИЯ СВОЙСТВА ЖИВОСТИ (NS)

```
gcc -DMEMLIM=24576 -O2 -DCOLLAPSE -DXUSAFE -w -o pan pan.c
./pan -m10000000  -a -n -N liveness0
Pid: 4556
pan: ltl formula liveness0
Depth=     223 States=    1e+06 Transitions= 5.18e+06 Memory=   692.038
     t=     1.22 R=   8e+05
                                        …
Depth=   54296 States= 5.11e+08 Transitions= 3.98e+09 Memory= 21689.878
     t= 2.13e+03 R=   2e+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 104 byte, depth reached 54296, errors: 0
3.8177513e+08 states, stored (5.11863e+08 visited)
3.4696359e+09 states, matched
3.9814986e+09 transitions (= visited+matched)
        7 atomic steps
hash conflicts: 4.9895574e+09 (resolved)

Stats on memory usage (in Megabytes):
48059.766  equivalent memory usage for states (stored*(State-vector +
overhead))
20696.161  actual memory usage for states (compression: 43.06%)
           state-vector as stored = 29 byte + 28 byte overhead
  512.000  memory used for hash table (-w26)
  534.058  memory used for DFS stack (-m10000000)
    6.247  memory lost to fragmentation
21735.972  total actual memory usage


nr of templates: [ 0:globals 1:chans 2:procs ]
collapse counts: [ 0:2658271 2:7 3:8 4:17 5:15 6:15 7:13 8:13 9:13 10:2
17:2 ]

pan: elapsed time 2.14e+03 seconds
No errors found -- did you verify all claims?
```

ПРИЛОЖЕНИЕ D

ВЕРИФИКАЦИЯ СВОЙСТВА СПРАВЕДЛИВОСТИ (NS)

```
gcc -DMEMLIM=24576 -O2 -DCOLLAPSE -DXUSAFE -w -o pan pan.c
./pan -m10000000  -a -n -N fairness0
Pid: 1472
pan: ltl formula fairness0
Depth=   12746 States=    1e+06 Transitions= 4.76e+06 Memory=   720.652
     t=      1.3 R=   8e+05
                                        …
Depth=   54296 States= 2.75e+08 Transitions= 1.65e+09 Memory= 15390.855
     t=      831 R=   3e+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 104 byte, depth reached 54296, errors: 0
2.6368561e+08 states, stored (2.75684e+08 visited)
1.3764006e+09 states, matched
1.6520841e+09 transitions (= visited+matched)
        7 atomic steps
hash conflicts: 1.8021768e+09 (resolved)

Stats on memory usage (in Megabytes):
33194.066  equivalent memory usage for states (stored*(State-vector +
overhead))
14385.666  actual memory usage for states (compression: 43.34%)
           state-vector as stored = 29 byte + 28 byte overhead
  512.000  memory used for hash table (-w26)
  534.058  memory used for DFS stack (-m10000000)
    4.345  memory lost to fragmentation
15427.378  total actual memory usage


nr of templates: [ 0:globals 1:chans 2:procs ]
collapse counts: [ 0:2658271 2:7 3:8 4:17 5:15 6:15 7:13 8:13 9:13 10:2
23:2 ]

pan: elapsed time 834 seconds
No errors found -- did you verify all claims?
```