Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Факультет информационных технологий и программирования

Кафедра «Компьютерные Технологии»

Отчет по лабораторной работе

«Верификация программ»

Выполнил студент гр. 3539 А.Д. Богуцкий

Санкт-Петербург

2011

Оглавление

Введение

В лабораторной работе требуется выполнить задания по верификаторам **SPIN** и **NuSMV**. Для выполнения задания использовались графический пользовательский интерфейс jSpin для верификатора SPIN и верификатор NuSMV 2.5.2 с текстовым интерфейсом.

1. Постановка задач

Для верификатора SPIN требовалось построить автомат Бюхи по выданным LTL-формулам и исправить данную программу.

Для верификатора NuSMV требовалось смоделировать один из данных алгоритмов параллельного программирования и, с помощью CTL, проверить отсутствие голодания и наличие взаимного исключения критических секций для данного алгоритма.

2. Решение задач

2.1. Верификатор SPIN

2.1.1. Построить автомат Бюхи по темпоральным формулам G(p->Fq) и G(p U q)

Построим автомат Бюхи для формулы G(p->Fq) с помощью SPIN, получаем такое описание автомата:

```
T0_init:
     if
     :: (((! ((p))) || ((q)))) -> goto accept_S20
     :: (1) -> goto T0_S27
     fi;
accept_S20:
     if
     :: (((! ((p))) || ((q)))) -> goto T0_init
     :: (1) -> goto T0_S27
     fi;
accept_S27:
     if
     :: ((q)) -> goto T0_init
     :: (1) -> goto T0_S27
     fi;
T0_S27:
     if
     :: ((q)) -> goto accept_S20
     :: (1) -> goto T0_S27
     :: ((q)) -> goto accept_S27
     fi;
```

Суффикс Init означает, что состояние стартовое. Префикс Accept, означает, что состояние допускающее. Автомат в графическом виде представлен на рис. 1.

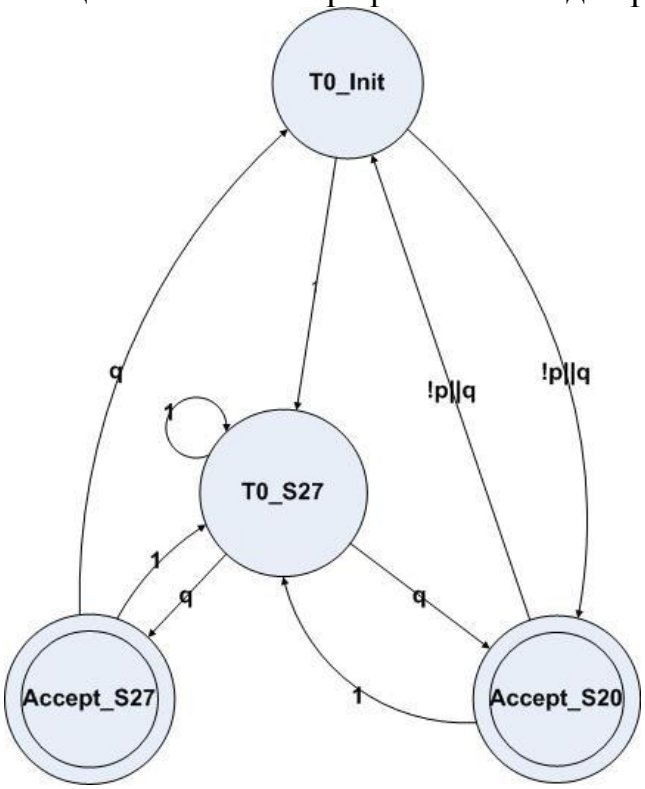

Рис. 1 – Автомат Бюхи по версии jSPIN для формулы G (p -> Fq)

Теперь устраним лишние вершины:

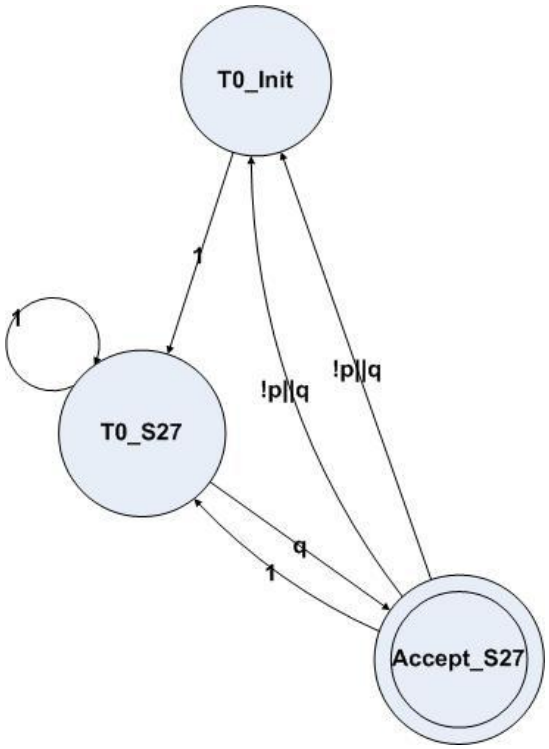

Рис. 2 – Автомат Бюхи для формулы G (p -> Fq)

Автомат Бюхи для формулы G (p -> Fq) построен.

Построим автомат Бюхи для формулы G(pUq) с помощью SPIN, получаем такое описание автомата:

```
T0_init:
     if
     :: ((q)) -> goto accept_S9
     :: ((p)) -> goto T0_init
     fi;
accept_S9:
     if
     :: ((((p)) || ((q)))) -> goto T0_init
     fi;
```

Представим автомат в графическом виде.

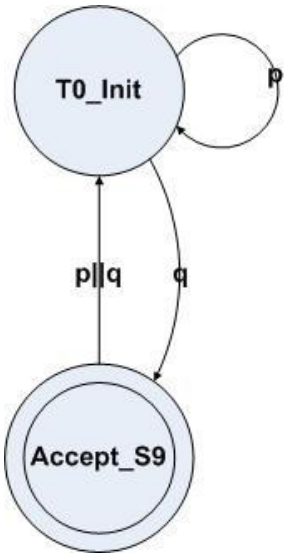

Рис.3 – Автомат Бюхи для формулы G (p U q)

2.1.2. Исправить данную PROMELA-спецификацию:

```
bit X, Y;
proctype C(){
     do
     :: true ->  X = 0; Y = X
     :: true ->  X = 1; Y = X
     od
}
proctype monitor(){
     do
     :: assert(X==Y)
     od

}
init{ atomic { run C(); run monitor() } }
```

чтобы утверждение assert(X==Y) не нарушалось и объясните результат:

В исходной спецификации программа выдает assertion violated (X==Y). Это происходит, так как операции присваивания происходит не атомарно и поэтому процесс C может прерваться, а процесс monitor вызваться. В таком случае X не будет равен Y. Поэтому, чтобы исправить спецификацию надо задать операции X=0; Y=X и X=1; Y=X атомарными. В такой спецификации утверждение assert (X==Y) не будет нарушаться. Далее приведен код работающей программы.

```
bit X, Y;
proctype C(){
     do
     :: true ->  atomic { X = 0; Y = X }
     :: true ->  atomic { X = 1; Y = X }
     od
}
proctype monitor(){
     do
     :: assert(X==Y)
     od
}
init{
     atomic {
          run C();
          run monitor()
     }
}
```

2.2. Верификатор NuSMV. Реализовать в спецификации NuSMV алгоритм Петерсона. Провести его тестирование для двух, трех, четырех процессов. Проверить, что он удовлетворяет свойствам взаимного исключения и отсутствия голодания.

Алгоритм Петерсона в псевдокоде:

```
While true do begin
     For j:=1 to N-1 do begin
          Queue[i]:=j
          Turn[j]:=i;
          Wait until (\forall k != i: Queue[k]<j) or Turn[j] != i
     End;
<critical section>
Queue[i] := 0
     End
```

Данная программа моделирует алгоритм Петерсона для четырех процессов:

```
MODULE proc(queue,turn,i)
VAR
     j: 1..4;
     state: {enter,waiting,critical,exit};
ASSIGN
     init(j):= 1;
     init(state) := enter;
     next(state) := case
          state = enter : waiting;
          state = critical : exit;
          state = exit : enter;
          state = waiting & next(j) = 4 : critical;
          TRUE : waiting;
     esac;
     next(turn[1]) := case
          next(j) = 1 : i;
          TRUE : turn[1];
     esac;
     next(turn[2]) := case
          next(j) = 2 : i;
          TRUE : turn[2];
     esac;
     next(turn[3]) := case
          next(j) = 3 : i;
          TRUE : turn[3];
     esac;
```

```
     next(turn[4]) := case
          next(j) = 4 : i;
          TRUE : turn[4];
     esac;
     next(queue[i]) := case
          next(state) = exit : 0;
          next(state) = waiting : next(j);
          TRUE : queue[i];
     esac;
     next(j) := case
          state = exit : 1;
          state = waiting & (((queue[1] < j | i = 1) &
              (queue[2] < j | i = 2) &
              (queue[3] < j | i = 3) &
              (queue[4] < j | i = 4)) | turn[j] != i) :
              j = 4 ? 4 : j + 1;
          TRUE : j;
     esac;

FAIRNESS
     running

MODULE main
VAR
     queue : array 1..4 of 0..4;
     turn : array 1..4 of 0..4;
     proc1 : process proc(queue,turn,1);
     proc2 : process proc(queue,turn,2);
     proc3 : process proc(queue,turn,3);
     proc4 : process proc(queue,turn,4);
ASSIGN
     init(queue[1]) := 0;
     init(queue[2]) := 0;
     init(queue[3]) := 0;
     init(queue[4]) := 0;
     init(turn[1]) := 1;
     init(turn[2]) := 1;
     init(turn[3]) := 1;
     init(turn[4]) := 1;

SPEC AG !((proc1.state = critical & proc2.state = critical) |
            (proc1.state = critical & proc3.state = critical) |
            (proc1.state = critical & proc4.state = critical) |
            (proc2.state = critical & proc3.state = critical) |
            (proc2.state = critical & proc4.state = critical) |
```

```
            (proc3.state = critical & proc4.state = critical))
SPEC AG (proc1.state = waiting -> AF proc1.state = critical)
SPEC AG AF proc1.state = critical
```

После проверки с помощью NuSMV мы убеждаемся, что свойства отсутствия голодания и взаимного исключения выполняются.

```
-  SPECIFICATION AG !((proc1.state = critical & proc2.state = critical) |
        (proc1.state = critical & proc3.state = critical) |
        (proc1.state = critical & proc4.state = critical) |
        (proc2.state = critical & proc3.state = critical) |
        (proc2.state = critical & proc4.state = critical) |
        (proc3.state = critical & proc4.state = critical)) is true
-  SPECIFICATION AG (proc1.state = waiting -> AF proc1.state = critical)
is true
-  SPECIFICATION AG AF proc1.state = critical is true
```

Алгоритм Петерсона для двух и трех потоков в NuSMV выполняется аналогично алгоритму для четырех потоков. Отличия заключаются в размерности массивов `queue` и *turn*, описании цикла `For j:=1 to N-1 do begin` (в коде NuSMV – смена состояния `j` и переход `state` из состояния `wait`), а так же в обработки отсутствующих элементов массивов `queue` и `turn`.

3. Выводы

Мною были успешно усвоены спецификации SPIN и NuSMV.

4. Список литературы

1. jSpin tutorial (прилагается к программе jSPIN*:* *http://code.google.com/p/jspin/*)
2. NuSMV tutorial: *http://nusmv.fbk.eu/NuSMV/tutorial/v25/tutorial.pdf*
3. NuSMV user manual: *http://nusmv.fbk.eu/NuSMV/userman/v25/nusmv.pdf*