МИНИСТЕРСТВА НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Курсовой проект

по курсу: «Верификация и анализ программ»

Выполнил студент гр. 3540901/81501 Селиверстов Я.А.

Руководитель, к.т.н. Ицыксон В. М.

Санкт-Петербург
2019

ОГЛАВЛЕНИЕ

I. ЛАБОРАТОРНАЯ РАБОТА № 1

1.1. Цель работы

Построить и визуализировать граф потока управления программы на языке программирования C#.

1.2. Ход работы

1.2.1. Определения

Абстрактное синтаксическое дерево — помеченное ориентированное дерево, в котором внутренние вершины сопоставлены с операторами языка программирования, а листья — с соответствующими операндами. Таким образом, листья являются пустыми операторами и представляют только переменные и константы.[1]

По модели АСД легко восстановить исходную программу с точностью до форматирования и комментариев.[2]

Граф потока управления (CFG) — в теории компиляции — множество всех возможных путей исполнения программы, представленное в виде графа.

Как и большинство языков программирования, C# имеет встроенные средства для построения AST дерева программы. С помощью встроенных в Visual Studio инструментов, данное дерево можно визуализировать.

1.3. Построение CFG

Построение CFG графа заключается в обходе дерева, и создания специальной структуры для дальнейшей обработки.

Визуализация графа осуществляется средствами Microsoft.Msagl.Drawing и Microsoft.Msagl.GraphViewerGdi.

Код программы построения CFG графа представлен на Листинге 1.

Листинг 1.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Microsoft.CodeAnalysis;
using Microsoft.CodeAnalysis.CSharp;
using Microsoft.CodeAnalysis.CSharp.Syntax;
using Microsoft.CodeAnalysis.MSBuild;
using Microsoft.CodeAnalysis.FlowAnalysis;
using Microsoft.Msagl.Drawing;
using System.IO;
using Microsoft.Msagl.GraphViewerGdi;
using System.Drawing;
using System.Collections;
```

```
namespace AST
{
    class Program
    {
        class Node
        {
            public string name;
            public string content;
            public Node cond;
            public bool is_cond_node = false;
            public Microsoft.Msagl.Drawing.Node node;
        }
        class Edge
        {
            public Node source;
            public Node dest;
            public string note;
            public Microsoft.Msagl.Drawing.Edge edge;
        }
        class Graph
        {
            public ArrayList nodes;
            public ArrayList edges;
            public Dictionary<UInt64,Node> id2node;
            public Graph()
            {
                nodes = new ArrayList();
                edges = new ArrayList();
                id2node = new Dictionary<ulong, Node>();
            }
        }
        static void Main(string[] args)
        {
            if (args.Length != 1)
            {
                System.Console.WriteLine("Usage:\n[progname] [source.cs]");
                return;
            }
            string source = File.ReadAllText(args[0], Encoding.UTF8);

            CSharpParseOptions options = CSharpParseOptions.Default.WithFeatures(new[]
{ new KeyValuePair<string, string>("flow-analysis", "") });

            var tree = CSharpSyntaxTree.ParseText(source, options);
            var compilation = CSharpCompilation.Create("c", new[] { tree });
            var model = compilation.GetSemanticModel(tree, ignoreAccessibility: true);
            Graph g = new Graph();
            UInt64 bid = 0;
```

```
        foreach (var methodBodySyntax in
tree.GetCompilationUnitRoot().DescendantNodes().OfType<BaseMethodDeclarationSyn
tax>()) {

            //   var methodBodySyntax =
tree.GetCompilationUnitRoot().DescendantNodes().OfType<BaseMethodDeclarationSyn
tax>().Last();
            string procname = methodBodySyntax.ToString().Substring(0,
methodBodySyntax.ToString().IndexOf("{"));
            Console.WriteLine(procname);
            var cfgFromSyntax = ControlFlowGraph.Create(methodBodySyntax, model);
            Dictionary<BasicBlock, UInt64> block_ids = new Dictionary<BasicBlock,
ulong>();

            foreach (var bb in cfgFromSyntax.Blocks)
            {
               block_ids[bb] = bid;

               Node n = new Node();
               n.name = String.Format("block_{0}:", bid);
               n.content = "";
               foreach (var op in bb.Operations)
               {
                  n.content += op.Syntax.ToString()+"\n";
               }
               g.nodes.Add(n);
               g.id2node[bid] = n;
               bid += 1;
               if (bb.ConditionKind != ControlFlowConditionKind.None &&
bb.BranchValue != null)
               {
                  Node n1 = new Node();
                  n1.content = bb.BranchValue.Syntax.ToString();
                  n1.is_cond_node = true;
                  n.cond = n1;
                  n1.name = String.Format("block_{0}:", bid);
                  g.nodes.Add(n1);
                  g.id2node[bid] = n1;
                  bid += 1;

               }
            }
            bool is_first = true;
            foreach (var bb in cfgFromSyntax.Blocks)
            {
               var x = block_ids[bb];
               System.Console.WriteLine(String.Format("block_{0}:", x));
               if (is_first)
```

```
                {
                    is_first = false;
                    g.id2node[x].content = "method "+procname + "\n"+g.id2node[x].content;
                }
                foreach (var op in bb.Operations) {
                    System.Console.WriteLine(op.Syntax.ToString());
                }

                if (bb.ConditionKind == ControlFlowConditionKind.None)
                {
                    if (bb.BranchValue != null)
                    {
                        g.id2node[x].content = g.id2node[x].content + "\n" + "return " +
bb.BranchValue.Syntax.ToString();
                        System.Console.WriteLine("return
"+bb.BranchValue.Syntax.ToString());
                    }
                    if (bb.FallThroughSuccessor != null)
                    {
                        Edge e = new Edge();
                        e.source = g.id2node[x];
                        var next = block_ids[bb.FallThroughSuccessor.Destination];
                        e.dest = g.id2node[next];
                        g.edges.Add(e);
                        System.Console.WriteLine(String.Format("block_{0}",
block_ids[bb.FallThroughSuccessor.Destination]));
                    }
                }
                else if (bb.ConditionKind == ControlFlowConditionKind.WhenFalse)
                {
                    if (bb.BranchValue != null)
                    {
                        //g.id2node[x].content = g.id2node[x].content + "\n" +
bb.BranchValue.Syntax.ToString();
                        Edge e0 = new Edge();
                        e0.source = g.id2node[x];
                        e0.dest = g.id2node[x].cond;
                        g.edges.Add(e0);

                        Edge e1 = new Edge();
                        e1.source = e0.dest;
                        var next = block_ids[bb.ConditionalSuccessor.Destination];
                        e1.dest = g.id2node[next];
                        g.edges.Add(e1);

                        e1.note = "false";

                        Edge e2 = new Edge();
```

```
                        e2.source = e0.dest;
                        next = block_ids[bb.FallThroughSuccessor.Destination];
                        e2.dest = g.id2node[next];
                        g.edges.Add(e2);
                        e2.note = "true";
                        System.Console.WriteLine(bb.BranchValue.Syntax.ToString());
                    }//!!
                    else
                    {
                        Edge e1 = new Edge();
                        e1.source = g.id2node[x];
                        var next = block_ids[bb.ConditionalSuccessor.Destination];
                        e1.dest = g.id2node[next];
                        g.edges.Add(e1);
                        e1.note = bb.BranchValue.Syntax.ToString();
                        Edge e2 = new Edge();
                        e2.source = g.id2node[x];
                        next = block_ids[bb.FallThroughSuccessor.Destination];
                        e2.dest = g.id2node[next];
                        g.edges.Add(e2);
                        e2.note = "not " + bb.BranchValue.Syntax.ToString();
                      System.Console.WriteLine(String.Format("{0}?block_{1}:block_{2}",
bb.BranchValue.Syntax.ToString(),
                            block_ids[bb.ConditionalSuccessor.Destination],
block_ids[bb.FallThroughSuccessor.Destination]));
                    }
                }
                else
                {
                    if (bb.BranchValue != null)
                    {
                        Edge e0 = new Edge();
                        e0.source = g.id2node[x];
                        e0.dest = g.id2node[x].cond;
                        g.edges.Add(e0);

                        Edge e1 = new Edge();
                        e1.source = e0.dest;
                        var next = block_ids[bb.ConditionalSuccessor.Destination];
                        e1.dest = g.id2node[next];
                        g.edges.Add(e1);
                        e1.note = "true";
                        Edge e2 = new Edge();
                        e2.source = e0.dest;
                        next = block_ids[bb.FallThroughSuccessor.Destination];
                        e2.dest = g.id2node[next];
                        g.edges.Add(e2);
                        e2.note = "false";
```

```
                        System.Console.WriteLine(bb.BranchValue.Syntax.ToString());
                    }
                    else
                    {
                        Edge e1 = new Edge();
                        e1.source = g.id2node[x];
                        var next = block_ids[bb.ConditionalSuccessor.Destination];
                        e1.dest = g.id2node[next];
                        g.edges.Add(e1);
                        e1.note = "not " + bb.BranchValue.Syntax.ToString();
                        Edge e2 = new Edge();
                        e2.source = g.id2node[x];
                        next = block_ids[bb.FallThroughSuccessor.Destination];
                        e2.dest = g.id2node[next];
                        g.edges.Add(e2);
                        e2.note = bb.BranchValue.Syntax.ToString();
System.Console.WriteLine(String.Format("!{0}?block_{1}:block_{2}",
bb.BranchValue.Syntax.ToString(),
                            block_ids[bb.ConditionalSuccessor.Destination],
block_ids[bb.FallThroughSuccessor.Destination]));
                    }
                }
            }
        }
        Microsoft.Msagl.Drawing.Graph draw_graph = new
Microsoft.Msagl.Drawing.Graph("");
        /*foreach(var nd in g.nodes)
        {
            Microsoft.Msagl.Drawing.Node n = new
Microsoft.Msagl.Drawing.Node(((Node)nd).name);
            n.LabelText = ((Node)nd).content;
            draw_graph.AddNode(n);
            ((Node)nd).node = n;
        }
        foreach(var ed in g.edges)
        {
            Edge cur_e = (Edge)ed;
            var src = cur_e.source.node;
            var dst = cur_e.dest.node;
            Microsoft.Msagl.Drawing.Edge e = new Microsoft.Msagl.Drawing.Edge(src,
dst, ConnectionToGraph.Connected);
            draw_graph.AddEdge(src.Id, ((Edge)ed).note, dst.Id);
        }*/
        //!
        foreach (var ed in g.edges)
        {
            if (((Edge)ed).source.node == null)
            {
```

}

```
                string src_s = ((Edge)ed).source.name + "\n" + ((Edge)ed).source.content;
                Microsoft.Msagl.Drawing.Node n = new
Microsoft.Msagl.Drawing.Node(src_s);
                if (((Edge)ed).source.is_cond_node)
                    n.Attr.Shape = Shape.Diamond;
                draw_graph.AddNode(n);
                ((Edge)ed).source.node = n;
            }
            if (((Edge)ed).dest.node == null)
            {
                string src_s = ((Edge)ed).dest.name + "\n" + ((Edge)ed).dest.content;
                Microsoft.Msagl.Drawing.Node n = new
Microsoft.Msagl.Drawing.Node(src_s);
                if (((Edge)ed).dest.is_cond_node)
                    n.Attr.Shape = Shape.Diamond;
                draw_graph.AddNode(n);
                ((Edge)ed).dest.node = n;
            }
        }//!!
        foreach (var ed in g.edges)
        {
            string src_s = ((Edge)ed).source.name+ "\n" + ((Edge)ed).source.content;
            string edge_s = ((Edge)ed).note;
            if (edge_s == null)
            {
                edge_s = "";
            }
            string dest_s = ((Edge)ed).dest.name + "\n" + ((Edge)ed).dest.content;

            draw_graph.AddEdge(src_s,edge_s,dest_s);
//              break;
        }
        Microsoft.Msagl.GraphViewerGdi.GraphRenderer renderer = new
Microsoft.Msagl.GraphViewerGdi.GraphRenderer(draw_graph);
        renderer.CalculateLayout();
        int width = (int)(draw_graph.Width);
        Bitmap bitmap = new Bitmap(width, (int)(draw_graph.Height * (width /
draw_graph.Width)));
        renderer.Render(bitmap);
        bitmap.Save("test.png");
    }
  }
}
```

Код тестовой программы, для построения CFG граф дан на Листинге 2..

Листинг 2.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.CodeAnalysis;
using Microsoft.CodeAnalysis.CSharp;

namespace TopLevel
{
  class C
{
    int N(int y){
        if (y%2==0){
            return y/2;
        }
        return y*2;
    }
    int M(int x)
    {
      x = 0;
      int y = x * 3;
      for (int i=0; i<10; i++){
        y+=x;
        if (y>100){
            x=-1;
        }
        else if (y<10){
            x=N(x);
        }
        else{
            break;
        }
      }
      return y;
    }
}
}
```

1.4. Результаты построения

Для анализируемой программы результат вывода блоков CFG графа представлен на рисунке 1.1.

```
C:\Ярослав\Политех\AST2\AST\AST\bin\Debug>AST.exe 1.cs
int N(int y)
block_0:
block_1
block_1:
y%2==0
block_3:
return y/2
block_5
block_4:
return y*2
block_5
block_5:
int M(int x)

block_6:
block_7
block_7:
x = 0;
y = x * 3
block_8
block_8:
i=0
block_9
block_9:
i<10
block_11:
y+=x;
y>100
block_13:
x=-1;
block_17
block_14:
y<10
block_16:
x=N(x);
block_17
block_17:
i++
block_9
block_18:
return y
block_19
block_19:

C:\Ярослав\Политех\AST2\AST\AST\bin\Debug>
```

Рис.1.1.

Визуализация графа потока управления средствами библиотеки Microsoft Automatic Graph Layout для анализируемой программы представлена на рисунке 1.2.

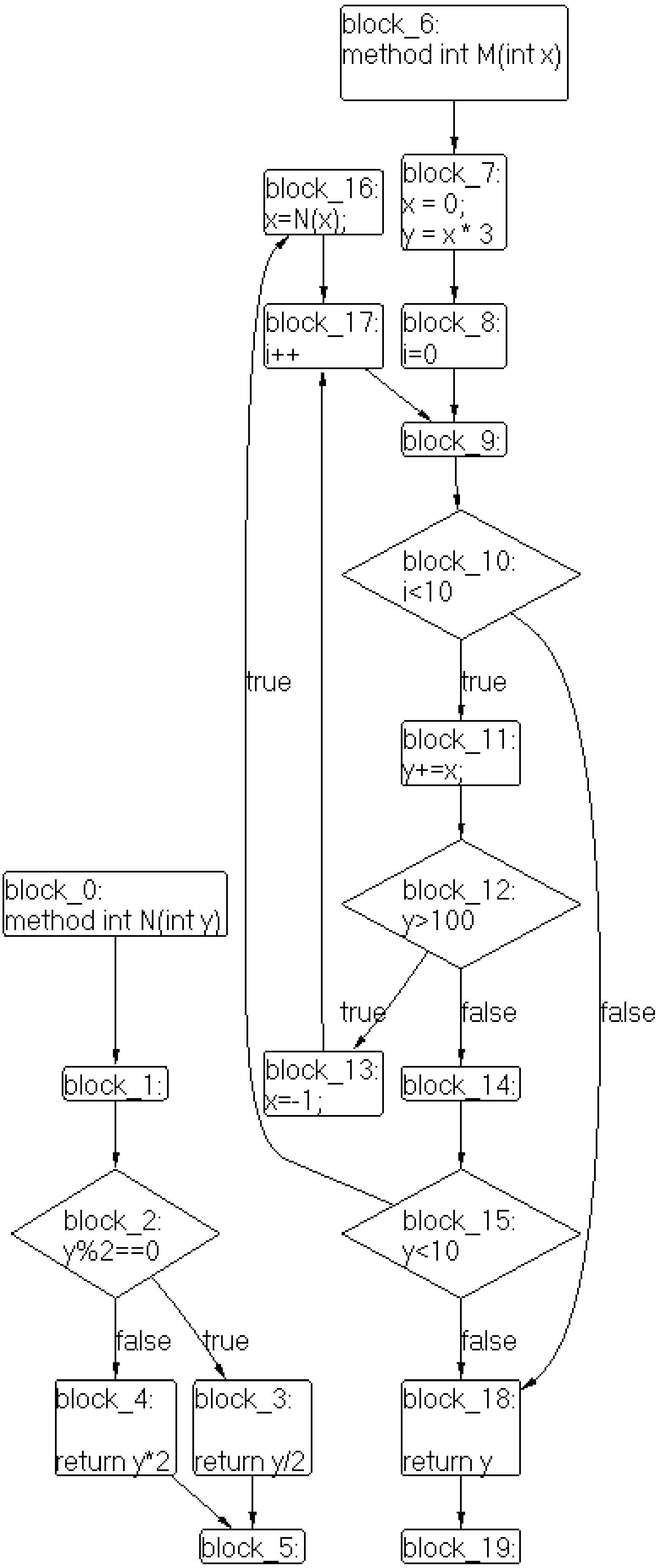

block_6:
method int M(int x)
block_7:
x = 0;
y = x * 3
block_16:
x=N(x);
block_17:
i++
block_8:
i=0
block_9:
block_10:
i<10
true
true
block_11:
y+=x;
block_0:
method int N(int y)
block_12:
y>100
true
false
false
block_1:
block_13:
x=-1;
block_14:
block_2:
y%2==0
block_15:
y<10
false
true
false
block_4:
return y*2
block_3:
return y/2
block_18:
return y
block_5:
block_19:

Рис.1.2.

1.5. ВЫВОД

В данной работе были получены навыки по построению CFG программы графа на языке C#. Благодаря разработанному подходу к анализу дерева, имеется возможность обрабатывать более сложные программы, потребуется лишь добавление логики обработки новых типов элементов.

II. ЛАБОРАТОРНАЯ РАБОТА №2.

2.1. Цель работы и условие задачи.

Необходимо провести верификацию модели при помощи NuSMV[4] для следующей системы:

Есть 5 пассажиров, стоящие на двух разных остановках. Автобус с тремя дверьми и двумя терминалами для оплаты едет по маршруту через 3 остановки. Составляющие модели изображены на рисунке 2.1.

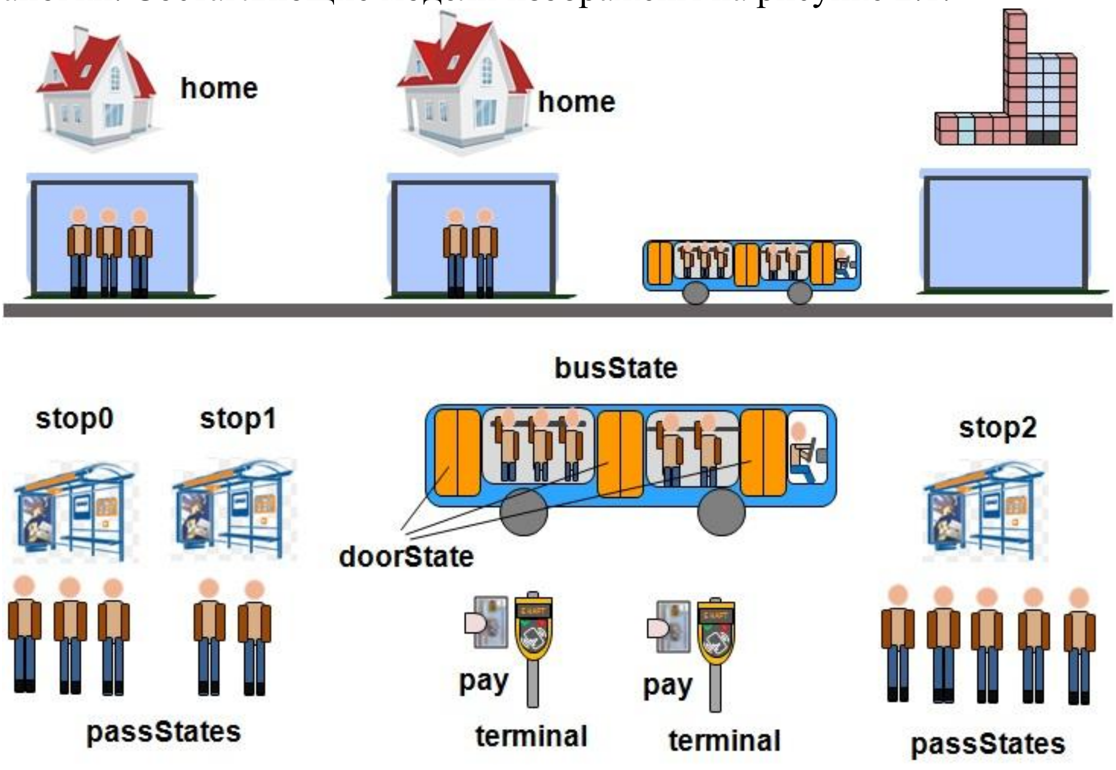


Рис.2.1.

Пассажиры входят в автобус и выходят из него на нужных для них остановках. Войдя в салон, пассажиры сразу оплачивают проезд в двух терминалах.

Этапы пассажира:

- Ожидание на остановке
- Вход в автобус
- Ожидание в очереди к терминалу
- Оплата проезда
- Проезд
- Выход из автобуса
- Отправление домой

Спецификации.

Необходимо проверить утверждения:

1) Автобус проедет через все остановки

2) Все кто зашел - оплатил проезд

3) Все окажутся дома

4) Все, кто заплатил – доедут

2.2. Ход работы

2.2.1.Модель.

Состояние пассажира

- Wait – ожидание на автобусной остановке
- Inn – вход в автобус
- payWait – ожидание оплаты в очереди к терминалу
- payment – оплата проезда
- drive – поездка на автобусе
- outt – выход из автобуса
- home – отправление домой с остановки

Состояния дверей автобуса

- allFree – все свободны
- twoFree – двое свободны
- oneFree – один свободен
- allBusy – все заняты

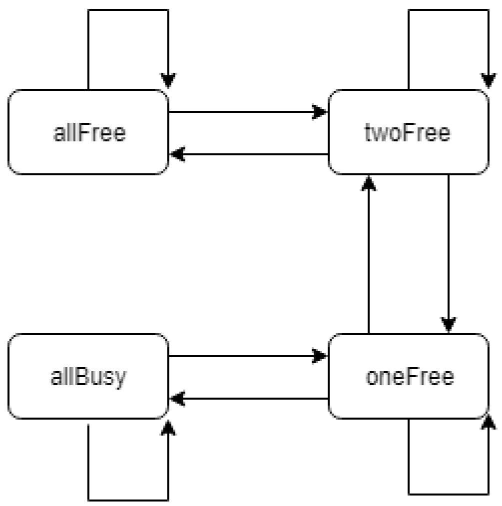


Рис.2.2.

Состояния открытости дверей автобуса

- Opened – открыты
- closed – закрыты

Состояния терминалов

- allBusy – оба заняты
- oneFree – один свободен
- allFree – оба свободны

Состояния автобуса

- Tr0 – подъезжание к первой остановке
- Stop0 – ожидание на остановке 1
- Tr1 – путь ко второй остановке
- Stop1 – ожидание на остановке 2
- Tr2 – путь к третей остановке
- Stop2 – ожидание на остановке 3
- Tr3 – путь дальше

2.2.2. Программа

```
MODULE main
VAR
        pass0 : process passenger (0, passStates, doorState, doorOp, busState, terminal,
payState, passStop, passQ, metk);
   pass1 : process passenger (1, passStates, doorState, doorOp, busState, terminal,
payState, passStop, passQ, metk);
   pass2 : process passenger (2, passStates, doorState, doorOp, busState, terminal,
payState, passStop, passQ, metk);
   pass3 : process passenger (3, passStates, doorState, doorOp, busState, terminal,
payState, passStop, passQ, metk);
   pass4 : process passenger (4, passStates, doorState, doorOp, busState, terminal,
payState, passStop, passQ, metk);

   passStates : array 0..4 of {wait, inn, payWait, payment, drive, outt, home};
   passStop : array 0..4 of {stop0, stop1};
   passQ : array 0..4 of {stop1, stop2};
   doorState : {allBusy, oneFree, twoFree, allFree};
   doorOp : {closed, opened};
   busState : {tr0, stop0, tr1, stop1, tr2, stop2, tr3};
   terminal : {allBusy, oneFree, allFree};
   payState : array 0..4 of {null, pay, not};
   metk : array 0..2 of {-1,0,1,2,3,4,5};

ASSIGN

   init(doorState) := allBusy;
   init(terminal) := allFree;
   init(busState) := tr0;
   init(doorOp) := closed;
   init(passStop[0]) := stop0;
   init(passStop[1]) := stop1;
   init(passStop[2]) := stop0;
```

```
   init(passStop[3]) := stop1;
   init(passStop[4]) := stop0;
   init(passQ[0]) := stop1;
   init(passQ[1]) := stop2;
   init(passQ[2]) := stop1;
   init(passQ[3]) := stop2;
   init(passQ[4]) := stop2;
   init(metk[0]) := 3;
   init(metk[1]) := 4;
   init(metk[2]) := 3;

-- Автобус проедет через все остановки
   CTLSPEC AG (busState = tr0 -> AF (busState = tr3))

-- Все кто зашел - оплатил проезд
   CTLSPEC AG ((payState[0] = null -> AF (payState[0] = pay)) & (payState[1] = null -
> AF (payState[1] = pay)) & (payState[2] = null -> AF (payState[2] = pay)) &
(payState[3] = null -> AF (payState[3] = pay)) & (payState[4] = null -> AF (payState[4]
= pay)))

-- Все окажутся дома
   LTLSPEC G ((passStates[0] = wait -> F passStates[0] = home) & (passStates[1] = wait
-> F passStates[1] = home) & (passStates[2] = wait -> F passStates[2] = home) &
(passStates[3] = wait -> F passStates[3] = home) & (passStates[4] = wait -> F
passStates[4] = home))

--Все, кто заплатил - уйдут
   LTLSPEC G ((payState[0] = pay -> F (passStates[0] = home)) & (payState[1] = pay ->
F (passStates[1] = home)) & (payState[2] = pay -> F (passStates[2] = home)) &
(payState[3] = pay -> F (passStates[3] = home)) & (payState[4] = pay -> F (passStates[4]
= home)))

MODULE passenger(id, passStates, doorState, doorOp, busState, terminal, payState,
passStop, passQ, metk)
VAR
   razr : {tr, tr1, fls};

DEFINE
        state := passStates[id];
   paym := payState[id];
   stop := passStop[id];
   quit := passQ[id];
ASSIGN
        init (passStates[id]) := wait;
   init (payState[id]) := null;
   init (razr) := tr;

next(passStop[id]) :=
```

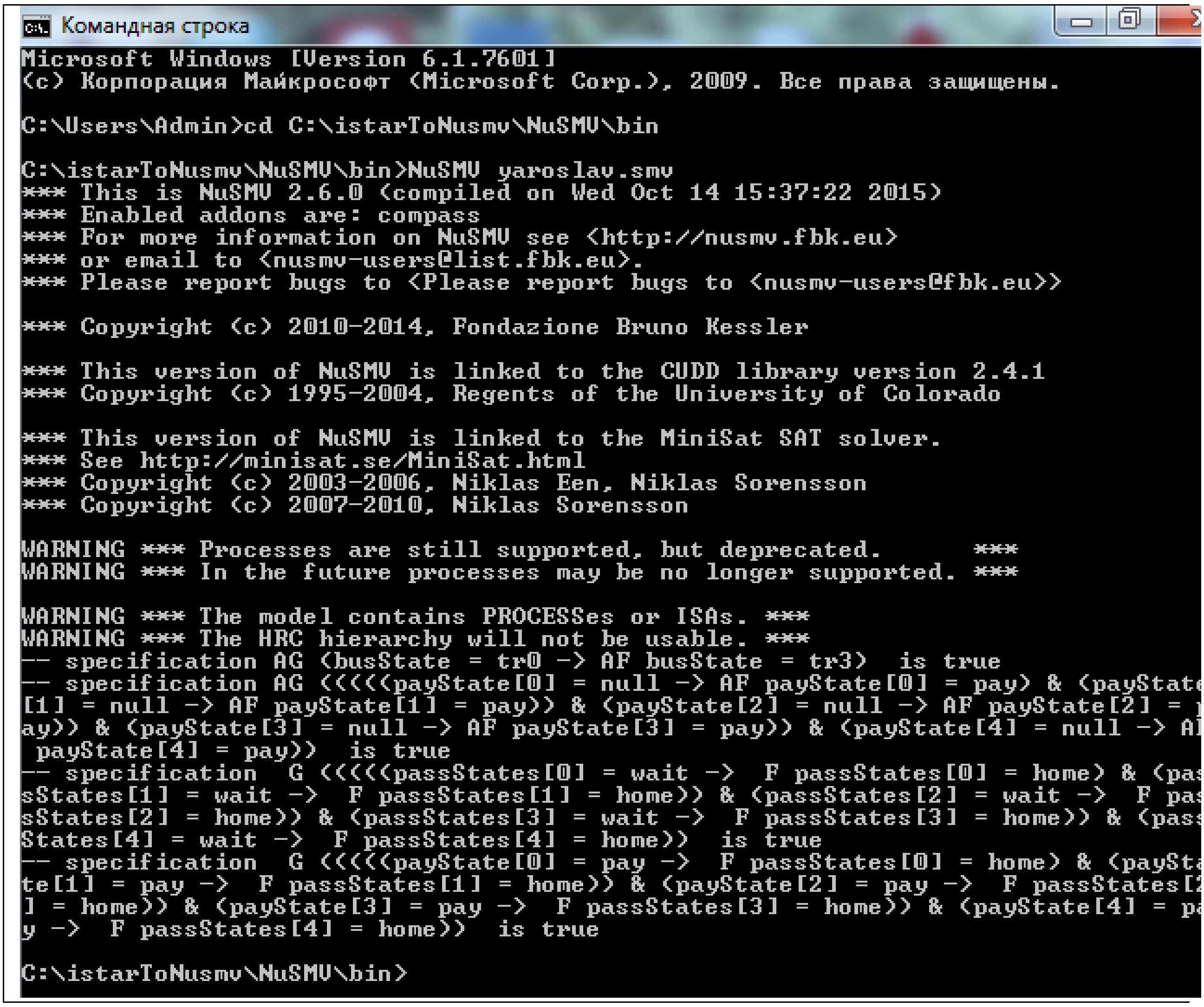
```
   case
      TRUE : passStop[id];
   esac;

next(passQ[id]) :=
   case
      TRUE : quit;
   esac;

next(doorState) :=
   case
      state = outt & doorState = allFree & doorOp = opened : twoFree;
      state = outt & doorState = twoFree & doorOp = opened : oneFree;
      state = outt & doorState = oneFree & doorOp = opened : allBusy;
      state = inn & doorState = allFree & doorOp = opened : twoFree;
      state = inn & doorState = twoFree & doorOp = opened : oneFree;
      state = inn & doorState = oneFree & doorOp = opened : allBusy;
      state = home & doorOp = opened & doorState = allBusy : oneFree;
      state = home & doorOp = opened & doorState = oneFree : twoFree;
      state = home & doorOp = opened & doorState = twoFree : allFree;
      state = drive & doorOp = opened & doorState = allBusy : oneFree;
      state = drive & doorOp = opened & doorState = oneFree : twoFree;
      state = drive & doorOp = opened & doorState = twoFree : allFree;
      TRUE : doorState;
   esac;

next(doorOp) :=
   case
      doorOp = closed & busState = stop0 : opened;
      doorOp = closed & busState = stop1 : opened;
      doorOp = closed & busState = stop2 : opened;
      doorOp = opened & doorState = allFree & busState = stop0 & metk[0] = 0 : closed;
      doorOp = opened & doorState = allFree & busState = stop1 & metk[1] = 0 : closed;
      doorOp = opened & doorState = allFree & busState = stop2 & metk[2] = 0 : closed;
      TRUE : doorOp;
   esac;

next(metk[0]) :=
   case
      busState = stop0 & stop = stop0 & state = drive & metk[0] = 5 & razr = tr : 4;
      busState = stop0 & stop = stop0 & state = drive & metk[0] = 4 & razr = tr : 3;
      busState = stop0 & stop = stop0 & state = drive & metk[0] = 3 & razr = tr : 2;
      busState = stop0 & stop = stop0 & state = drive & metk[0] = 2 & razr = tr : 1;
      busState = stop0 & stop = stop0 & state = drive & metk[0] = 1 & razr = tr : 0;
      TRUE : metk[0];
   esac;

next(razr) :=
```

```
   case
      state = drive : tr1;
      state = home : fls;
      TRUE : razr;
   esac;

next(metk[1]) :=
   case
      busState = stop1 & stop = stop1 & state = drive & metk[1] = 5 & razr = tr : 4;
      busState = stop1 & stop = stop1 & state = drive & metk[1] = 4 & razr = tr : 3;
      busState = stop1 & stop = stop1 & state = drive & metk[1] = 3 & razr = tr : 2;
      busState = stop1 & stop = stop1 & state = drive & metk[1] = 2 & razr = tr : 1;
      busState = stop1 & stop = stop1 & state = drive & metk[1] = 1 & razr = tr : 0;
      busState = stop1 & quit = stop1 & state = home & metk[1] = 5 & razr = tr1 : 4;
      busState = stop1 & quit = stop1 & state = home & metk[1] = 4 & razr = tr1 : 3;
      busState = stop1 & quit = stop1 & state = home & metk[1] = 3 & razr = tr1 : 2;
      busState = stop1 & quit = stop1 & state = home & metk[1] = 2 & razr = tr1 : 1;
      busState = stop1 & quit = stop1 & state = home & metk[1] = 1 & razr = tr1 : 0;
      TRUE : metk[1];
   esac;

next(metk[2]) :=
   case
      busState = stop2 & quit = stop2 & state = home & metk[2] = 5 & razr = tr1 : 4;
      busState = stop2 & quit = stop2 & state = home & metk[2] = 4 & razr = tr1 : 3;
      busState = stop2 & quit = stop2 & state = home & metk[2] = 3 & razr = tr1 : 2;
      busState = stop2 & quit = stop2 & state = home & metk[2] = 2 & razr = tr1 : 1;
      busState = stop2 & quit = stop2 & state = home & metk[2] = 1 & razr = tr1 : 0;
      TRUE : metk[2];
   esac;

next(busState) :=
   case
      busState = tr0 : stop0;
      busState = stop0 & metk[0] = 0 & doorOp = closed : tr1;
      busState = tr1 : stop1;
      busState = stop1 & metk[1] = 0 & doorOp = closed : tr2;
      busState = tr2 : stop2;
      busState = stop2 & metk[2] = 0 & doorOp = closed : tr3;
      TRUE : busState;
   esac;

next(terminal) :=
   case
      state = payWait & terminal = allFree : oneFree;
      state = payWait & terminal = oneFree : allBusy;
      state = drive & terminal = allBusy : oneFree;
      state = drive & terminal = oneFree : allFree;
```

```
      TRUE : terminal;
   esac;

next(payState[id]) :=
   case
      state = payment & paym = null : pay;
      TRUE : paym;
   esac;

next(passStates[id]) :=
   case
      state = wait & stop = stop0 & busState = stop0 & doorOp = opened : inn;
      state = wait & stop = stop1 & busState = stop1 & doorOp = opened & doorState !=
allBusy : inn;
      state = inn : payWait;
      state = payWait & terminal != allBusy : payment;
      state = payment : drive;
      state = drive & quit = stop1 & busState = stop1 & doorOp = opened & doorState !=
allBusy : outt;
      state = drive & quit = stop2 & busState = stop2 & doorOp = opened & doorState !=
allBusy : outt;
      state = outt : home;
      --state = home : wait;
      TRUE : state;
   esac;

FAIRNESS
  running
```

2.3. Результаты

Результаты моделирования представлены на рисунке 2.3. ниже.



```
Командная строка
Microsoft Windows [Version 6.1.7601]
(c) Корпорация Майкрософт (Microsoft Corp.), 2009. Все права защищены.

C:\Users\Admin>cd C:\istarToNusmv\NuSMV\bin

C:\istarToNusmv\NuSMV\bin>NuSMV yaroslav.smv
*** This is NuSMV 2.6.0 (compiled on Wed Oct 14 15:37:22 2015)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <Please report bugs to <nusmv-users@fbk.eu>>

*** Copyright (c) 2010-2014, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson

WARNING *** Processes are still supported, but deprecated.      ***
WARNING *** In the future processes may be no longer supported. ***

WARNING *** The model contains PROCESSes or ISAs. ***
WARNING *** The HRC hierarchy will not be usable. ***
-- specification AG (busState = tr0 -> AF busState = tr3)  is true
-- specification AG ((((payState[0] = null -> AF payState[0] = pay) & (payState
[1] = null -> AF payState[1] = pay)) & (payState[2] = null -> AF payState[2] = p
ay)) & (payState[3] = null -> AF payState[3] = pay)) & (payState[4] = null -> AF
 payState[4] = pay))  is true
-- specification  G ((((passStates[0] = wait ->  F passStates[0] = home) & (pas
sStates[1] = wait ->  F passStates[1] = home)) & (passStates[2] = wait ->  F pas
sStates[2] = home)) & (passStates[3] = wait ->  F passStates[3] = home)) & (pass
States[4] = wait ->  F passStates[4] = home))  is true
-- specification  G ((((payState[0] = pay ->  F passStates[0] = home) & (paySta
te[1] = pay ->  F passStates[1] = home)) & (payState[2] = pay ->  F passStates[2
] = home)) & (payState[3] = pay ->  F passStates[3] = home)) & (payState[4] = pa
y ->  F passStates[4] = home))  is true

C:\istarToNusmv\NuSMV\bin>
```

Рис.2.3.

На рисунке 2.3. изображен запуск программы с проверкой спецификаций.

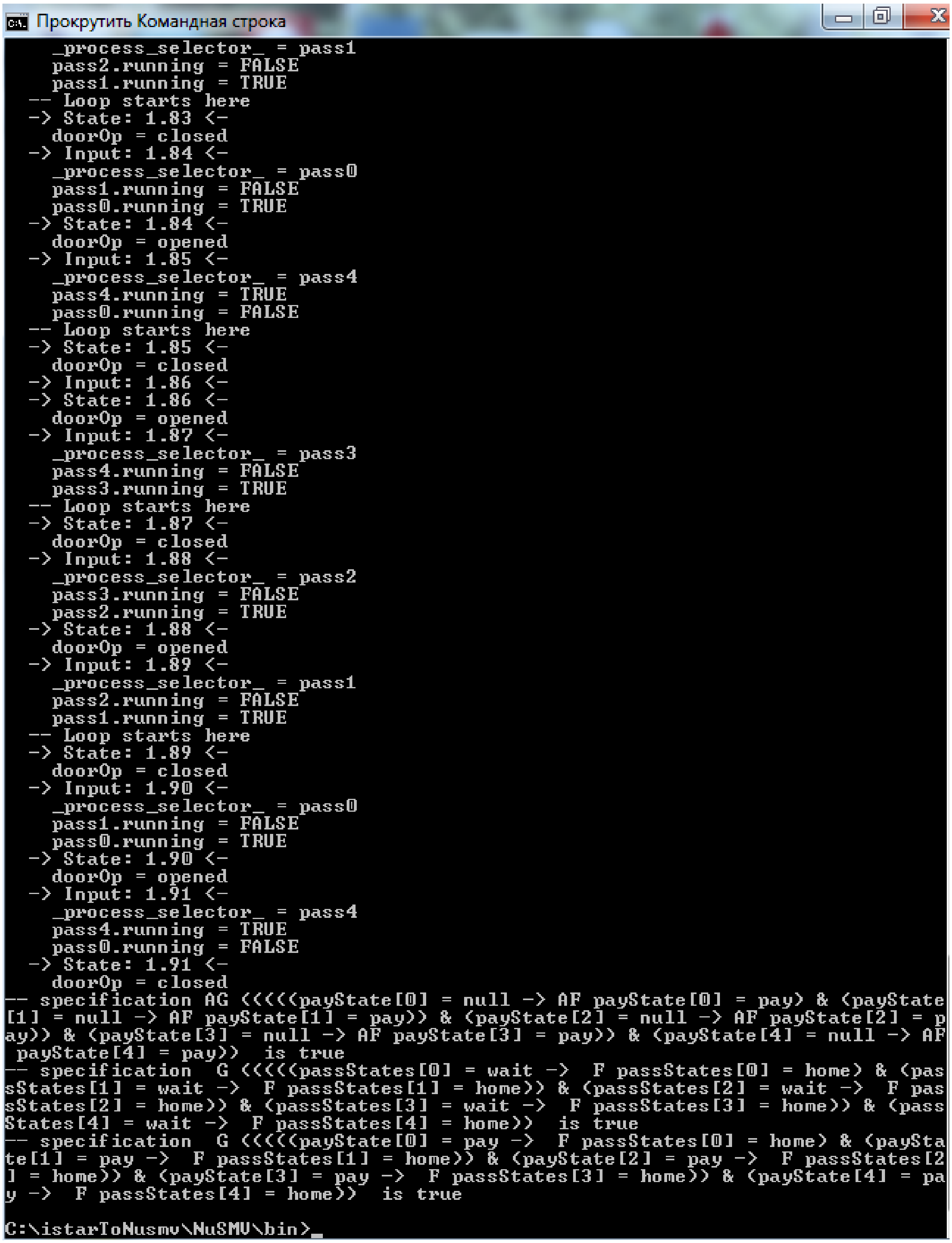
На следующем этапе произведем моделирование модели при нарушении каждой из спецификаций модели.

2.3.1. Нарушение первой спецификации.

<u>*1) Автобус проедет через все остановки*</u>

--159. busState = stop2 & metk[2] = 0 & doorOp = closed : tr3; // автобус остался стоять на третей остановке, никуда дальше не уезжая.

Результат вывода с нарушенной первой спецификацией, представлен на рисунке 2.4.



```
Прокрутить Командная строка
    _process_selector_ = pass1
    pass2.running = FALSE
    pass1.running = TRUE
  -- Loop starts here
  -> State: 1.83 <-
    doorOp = closed
  -> Input: 1.84 <-
    _process_selector_ = pass0
    pass1.running = FALSE
    pass0.running = TRUE
  -> State: 1.84 <-
    doorOp = opened
  -> Input: 1.85 <-
    _process_selector_ = pass4
    pass4.running = TRUE
    pass0.running = FALSE
  -- Loop starts here
  -> State: 1.85 <-
    doorOp = closed
  -> Input: 1.86 <-
  -> State: 1.86 <-
    doorOp = opened
  -> Input: 1.87 <-
    _process_selector_ = pass3
    pass4.running = FALSE
    pass3.running = TRUE
  -- Loop starts here
  -> State: 1.87 <-
    doorOp = closed
  -> Input: 1.88 <-
    _process_selector_ = pass2
    pass3.running = FALSE
    pass2.running = TRUE
  -> State: 1.88 <-
    doorOp = opened
  -> Input: 1.89 <-
    _process_selector_ = pass1
    pass2.running = FALSE
    pass1.running = TRUE
  -- Loop starts here
  -> State: 1.89 <-
    doorOp = closed
  -> Input: 1.90 <-
    _process_selector_ = pass0
    pass1.running = FALSE
    pass0.running = TRUE
  -> State: 1.90 <-
    doorOp = opened
  -> Input: 1.91 <-
    _process_selector_ = pass4
    pass4.running = TRUE
    pass0.running = FALSE
  -> State: 1.91 <-
    doorOp = closed
-- specification AG ((((( payState[0] = null -> AF payState[0] = pay) & (payState
[1] = null -> AF payState[1] = pay)) & (payState[2] = null -> AF payState[2] = p
ay)) & (payState[3] = null -> AF payState[3] = pay)) & (payState[4] = null -> AF
 payState[4] = pay))  is true
-- specification  G (((((passStates[0] = wait ->  F passStates[0] = home) & (pas
sStates[1] = wait ->  F passStates[1] = home)) & (passStates[2] = wait ->  F pas
sStates[2] = home)) & (passStates[3] = wait ->  F passStates[3] = home)) & (pass
States[4] = wait ->  F passStates[4] = home))  is true
-- specification  G (((((payState[0] = pay ->  F passStates[0] = home) & (paySta
te[1] = pay ->  F passStates[1] = home)) & (payState[2] = pay ->  F passStates[2
] = home)) & (payState[3] = pay ->  F passStates[3] = home)) & (payState[4] = pa
y ->  F passStates[4] = home))  is true

C:\istarToNusmv\NuSMV\bin>_
```

Рис.2.4.

2.3.2. Нарушение второй спецификации.

2) Все кто зашел - оплатил проезд

174. state = payment & paym = null : not; // Никто из пассажиров не оплатил проезд.

Результат вывода с нарушенной второй спецификацией, представлен на рисунке 2.5.

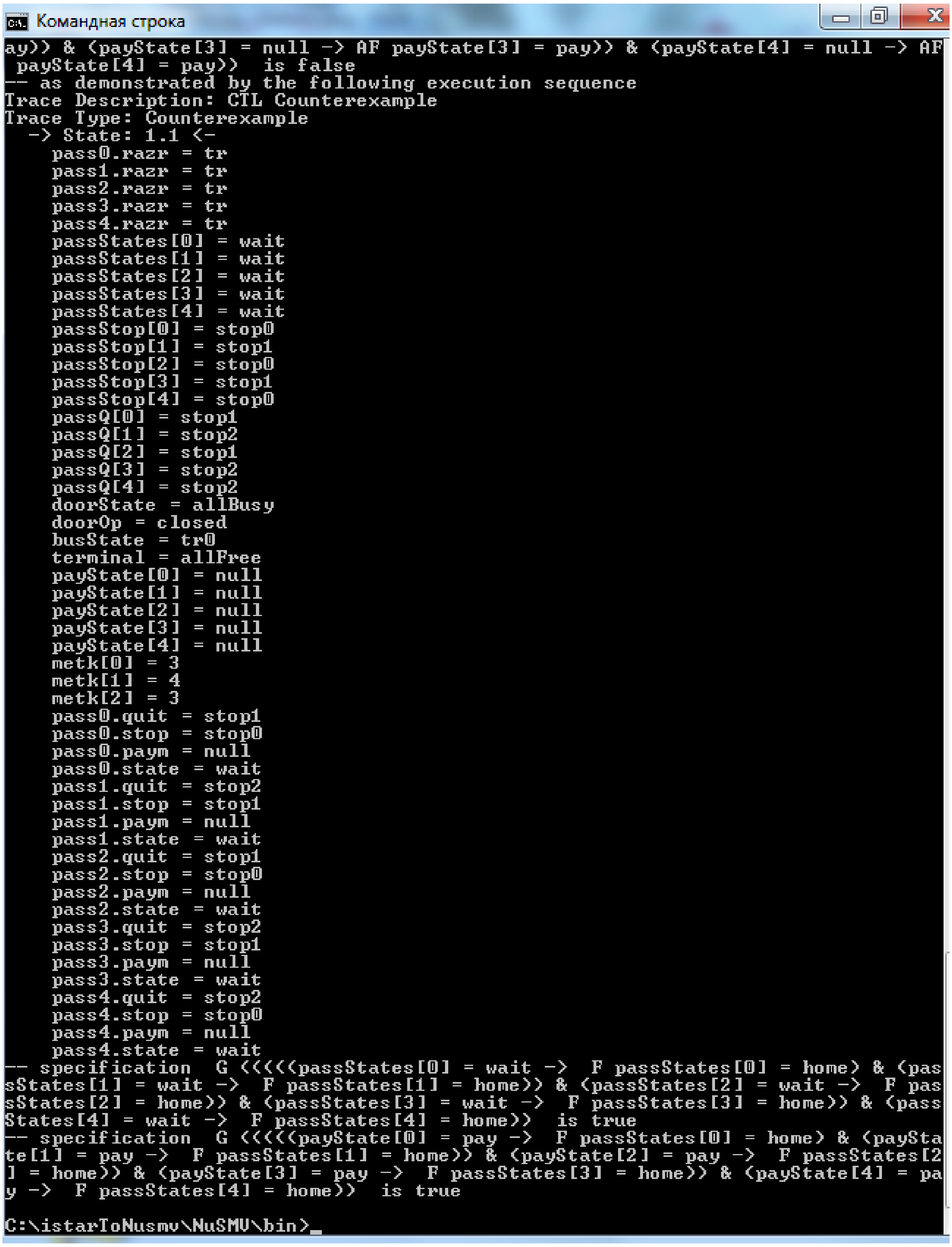


Рис.2.5.

2.3.3. Нарушение третьей и четвертой спецификации.

<u>*3) Все окажутся дома*</u>
<u>*4) Все, кто заплатил – доедут*</u>

Пассажиры передумали идти домой и остались ждать следующего автобуса.

188. state = home : wait; // Раскоментировать строчку 188. Пассажиры передумали идти домой и остались ждать следующего автобуса.

Результат вывода с нарушенной третьей и четвертой спецификацией, представлен на рисунке 2.6.

```
Командная строка
    busState = tr3
  -> Input: 2.68 <-
    _process_selector_ = pass1
    pass1.running = TRUE
    pass0.running = FALSE
  -- Loop starts here
  -> State: 2.68 <-
  -> Input: 2.69 <-
    _process_selector_ = pass2
    pass2.running = TRUE
    pass1.running = FALSE
  -- Loop starts here
  -> State: 2.69 <-
  -> Input: 2.70 <-
    _process_selector_ = pass3
    pass3.running = TRUE
    pass2.running = FALSE
  -- Loop starts here
  -> State: 2.70 <-
  -> Input: 2.71 <-
    _process_selector_ = pass4
    pass4.running = TRUE
    pass3.running = FALSE
  -- Loop starts here
  -> State: 2.71 <-
  -> Input: 2.72 <-
    _process_selector_ = main
    running = TRUE
    pass4.running = FALSE
  -- Loop starts here
  -> State: 2.72 <-
```

```
  -> Input: 2.73 <-
    _process_selector_ = pass0
    running = FALSE
    pass0.running = TRUE
  -- Loop starts here
  -> State: 2.73 <-
  -> Input: 2.74 <-
    _process_selector_ = pass1
    pass1.running = TRUE
    pass0.running = FALSE
  -- Loop starts here
  -> State: 2.74 <-
  -> Input: 2.75 <-
    _process_selector_ = pass2
    pass2.running = TRUE
    pass1.running = FALSE
  -- Loop starts here
  -> State: 2.75 <-
  -> Input: 2.76 <-
    _process_selector_ = pass3
    pass3.running = TRUE
    pass2.running = FALSE
  -- Loop starts here
  -> State: 2.76 <-
  -> Input: 2.77 <-
    _process_selector_ = pass4
    pass4.running = TRUE
    pass3.running = FALSE
  -- Loop starts here
  -> State: 2.77 <-
  -> Input: 2.78 <-
    _process_selector_ = main
    running = TRUE
    pass4.running = FALSE
  -> State: 2.78 <-

C:\istarToNusmv\NuSMV\bin>
```

Рис.2.6.

При обычной работе программы без изменений спецификации проходят успешно.

2.3.4. Нарушение второй спецификации. Параллельный режим.

1) Два терминала заняты обслуживанием двух пассажиров, третий пассажир не может оплатить вторая спецификация не выполняется.

```
MODULE main
VAR

        pass0 : process passenger (0, passStates, doorState, doorOp, busState, terminal,
payState, passStop, passQ, metk);
   pass1 : process passenger (1, passStates, doorState, doorOp, busState, terminal,
payState, passStop, passQ, metk);
   pass2 : process passenger (2, passStates, doorState, doorOp, busState, terminal,
payState, passStop, passQ, metk);
   pass3 : process passenger (3, passStates, doorState, doorOp, busState, terminal,
payState, passStop, passQ, metk);
   pass4 : process passenger (4, passStates, doorState, doorOp, busState, terminal,
payState, passStop, passQ, metk);

   passStates : array 0..4 of {wait, inn, payWait, payment, drive, outt, home};
   passStop : array 0..4 of {stop0, stop1};
   passQ : array 0..4 of {stop1, stop2};
   doorState : {allBusy, oneFree, twoFree, allFree};
   doorOp : {closed, opened};
   busState : {tr0, stop0, tr1, stop1, tr2, stop2, tr3};
   terminal : {allBusy, oneFree, allFree, allBusy2};
   payState : array 0..4 of {null, pay, not};
   metk : array 0..2 of {-1,0,1,2,3,4,5};

ASSIGN

   init(doorState) := allBusy;
   init(terminal) := allFree;
   init(busState) := tr0;
   init(doorOp) := closed;
   init(passStop[0]) := stop0;
   init(passStop[1]) := stop1;
   init(passStop[2]) := stop0;
   init(passStop[3]) := stop1;
   init(passStop[4]) := stop0;
   init(passQ[0]) := stop1;
   init(passQ[1]) := stop2;
   init(passQ[2]) := stop1;
```

```
   init(passQ[3]) := stop2;
   init(passQ[4]) := stop2;
   init(metk[0]) := 3;
   init(metk[1]) := 4;
   init(metk[2]) := 3;

-- Автобус проедет через все остановки
   CTLSPEC AG (busState = tr0 -> AF (busState = tr3))

-- Все кто зашел - оплатил проезд
   CTLSPEC AG ((payState[0] = null -> AF (payState[0] = pay)) & (payState[1] = null -
> AF (payState[1] = pay)) & (payState[2] = null -> AF (payState[2] = pay)) &
(payState[3] = null -> AF (payState[3] = pay)) & (payState[4] = null -> AF (payState[4]
= pay)))

-- Все окажутся дома
   LTLSPEC G ((passStates[0] = wait -> F passStates[0] = home) & (passStates[1] = wait
-> F passStates[1] = home) & (passStates[2] = wait -> F passStates[2] = home) &
(passStates[3] = wait -> F passStates[3] = home) & (passStates[4] = wait -> F
passStates[4] = home))

--Все, кто заплатил - уйдут
   LTLSPEC G ((payState[0] = pay -> F (passStates[0] = home)) & (payState[1] = pay ->
F (passStates[1] = home)) & (payState[2] = pay -> F (passStates[2] = home)) &
(payState[3] = pay -> F (passStates[3] = home)) & (payState[4] = pay -> F (passStates[4]
= home)))

MODULE passenger(id, passStates, doorState, doorOp, busState, terminal, payState,
passStop, passQ, metk)
VAR
   razr : {tr, tr1, fls};
DEFINE
        state := passStates[id];
   paym := payState[id];
   stop := passStop[id];
   quit := passQ[id];
ASSIGN
        init (passStates[id]) := wait;
   init (payState[id]) := null;
   init (razr) := tr;
next(passStop[id]) :=
   case
      TRUE : passStop[id];
   esac;
next(passQ[id]) :=
   case
      TRUE : quit;
   esac;
```

```
next(doorState) :=
   case
      state = outt & doorState = allFree & doorOp = opened : twoFree;
      state = outt & doorState = twoFree & doorOp = opened : oneFree;
      state = outt & doorState = oneFree & doorOp = opened : allBusy;
      state = inn & doorState = allFree & doorOp = opened : twoFree;
      state = inn & doorState = twoFree & doorOp = opened : oneFree;
      state = inn & doorState = oneFree & doorOp = opened : allBusy;
      state = home & doorOp = opened & doorState = allBusy : oneFree;
      state = home & doorOp = opened & doorState = oneFree : twoFree;
      state = home & doorOp = opened & doorState = twoFree : allFree;
      state = drive & doorOp = opened & doorState = allBusy : oneFree;
      state = drive & doorOp = opened & doorState = oneFree : twoFree;
      state = drive & doorOp = opened & doorState = twoFree : allFree;
      TRUE : doorState;
   esac;
next(doorOp) :=
   case
      doorOp = closed & busState = stop0 : opened;
      doorOp = closed & busState = stop1 : opened;
      doorOp = closed & busState = stop2 : opened;
      doorOp = opened & doorState = allFree & busState = stop0 & metk[0] = 0 : closed;
      doorOp = opened & doorState = allFree & busState = stop1 & metk[1] = 0 : closed;
      doorOp = opened & doorState = allFree & busState = stop2 & metk[2] = 0 : closed;
      TRUE : doorOp;
   esac;
next(metk[0]) :=
   case
      busState = stop0 & stop = stop0 & state = drive & metk[0] = 5 & razr = tr : 4;
      busState = stop0 & stop = stop0 & state = drive & metk[0] = 4 & razr = tr : 3;
      busState = stop0 & stop = stop0 & state = drive & metk[0] = 3 & razr = tr : 2;
      busState = stop0 & stop = stop0 & state = drive & metk[0] = 2 & razr = tr : 1;
      busState = stop0 & stop = stop0 & state = drive & metk[0] = 1 & razr = tr : 0;
      TRUE : metk[0];
   esac;
next(razr) :=
   case
      state = drive : tr1;
      state = home : fls;
      TRUE : razr;
   esac;
next(metk[1]) :=
   case
      busState = stop1 & stop = stop1 & state = drive & metk[1] = 5 & razr = tr : 4;
      busState = stop1 & stop = stop1 & state = drive & metk[1] = 4 & razr = tr : 3;
      busState = stop1 & stop = stop1 & state = drive & metk[1] = 3 & razr = tr : 2;
      busState = stop1 & stop = stop1 & state = drive & metk[1] = 2 & razr = tr : 1;
      busState = stop1 & stop = stop1 & state = drive & metk[1] = 1 & razr = tr : 0;
```

```
      busState = stop1 & quit = stop1 & state = home & metk[1] = 5 & razr = tr1 : 4;
      busState = stop1 & quit = stop1 & state = home & metk[1] = 4 & razr = tr1 : 3;
      busState = stop1 & quit = stop1 & state = home & metk[1] = 3 & razr = tr1 : 2;
      busState = stop1 & quit = stop1 & state = home & metk[1] = 2 & razr = tr1 : 1;
      busState = stop1 & quit = stop1 & state = home & metk[1] = 1 & razr = tr1 : 0;
      TRUE : metk[1];
   esac;
next(metk[2]) :=
   case
      busState = stop2 & quit = stop2 & state = home & metk[2] = 5 & razr = tr1 : 4;
      busState = stop2 & quit = stop2 & state = home & metk[2] = 4 & razr = tr1 : 3;
      busState = stop2 & quit = stop2 & state = home & metk[2] = 3 & razr = tr1 : 2;
      busState = stop2 & quit = stop2 & state = home & metk[2] = 2 & razr = tr1 : 1;
      busState = stop2 & quit = stop2 & state = home & metk[2] = 1 & razr = tr1 : 0;
      TRUE : metk[2];
   esac;

next(busState) :=
   case
      busState = tr0 : stop0;
      busState = stop0 & metk[0] = 0 & doorOp = closed : tr1;
      busState = tr1 : stop1;
      busState = stop1 & metk[1] = 0 & doorOp = closed : tr2;
      busState = tr2 : stop2;
      busState = stop2 & metk[2] = 0 & doorOp = closed : tr3;
      TRUE : busState;
   esac;
next(terminal) :=
   case
      state = payWait & terminal = allFree : oneFree;
      state = payWait & terminal = oneFree : allBusy;
      --state = payWait & terminal = allBusy : allBusy2;
      --state = drive & terminal = allBusy2 : allBusy;
      state = drive & terminal = allBusy : oneFree;
      state = drive & terminal = oneFree : allFree;
      TRUE : terminal;
   esac;
next(payState[id]) :=
   case
      state = payment & paym = null : pay;
      state = payWait & paym = null & terminal = allBusy : not;
      TRUE : paym;
   esac;
next(passStates[id]) :=
   case
      state = wait & stop = stop0 & busState = stop0 & doorOp = opened : inn;
      state = wait & stop = stop1 & busState = stop1 & doorOp = opened & doorState !=
allBusy : inn;
```

```
        state = inn : payWait;
        state = payWait & terminal != allBusy : payment;
        state = payment : drive;
        state = drive & quit = stop1 & busState = stop1 & doorOp = opened & doorState !=
allBusy : outt;
        state = drive & quit = stop2 & busState = stop2 & doorOp = opened & doorState !=
allBusy : outt;
        state = outt : home;
        --state = home : wait;
        TRUE : state;
    esac;
FAIRNESS
   running
```

Результат вывода программы, представлен на рисунке 2.7.

```
WARNING *** The model contains PROCESSes or ISAs. ***
WARNING *** The HRC hierarchy will not be usable. ***
-- specification AG (busState = tr0 -> AF busState = tr3)  is true
-- specification AG (((((payState[0] = null -> AF payState[0] = pay) & (payState
[1] = null -> AF payState[1] = pay)) & (payState[2] = null -> AF payState[2] = p
ay)) & (payState[3] = null -> AF payState[3] = pay)) & (payState[4] = null -> AF
 payState[4] = pay))  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    pass0.razr = tr
    pass1.razr = tr
    pass2.razr = tr
    pass3.razr = tr
    pass4.razr = tr
    passStates[0] = wait
    passStates[1] = wait
    passStates[2] = wait
    passStates[3] = wait
    passStates[4] = wait
    passStop[0] = stop0
    passStop[1] = stop1
    passStop[2] = stop0
    passStop[3] = stop1
    passStop[4] = stop0
    passQ[0] = stop1
    passQ[1] = stop2
    passQ[2] = stop1
    passQ[3] = stop2
    passQ[4] = stop2
    doorState = allBusy
    doorOp = closed
    busState = tr0
    terminal = allFree
    payState[0] = null
    payState[1] = null
    payState[2] = null
    payState[3] = null
    payState[4] = null
    metk[0] = 3
    metk[1] = 4
    metk[2] = 3
    pass0.quit = stop1
    pass0.stop = stop0
    pass0.paym = null
    pass0.state = wait
    pass1.quit = stop2
    pass1.stop = stop1
    pass1.paym = null
    pass1.state = wait
    pass2.quit = stop1
    pass2.stop = stop0
    pass2.paym = null
    pass2.state = wait
    pass3.quit = stop2
    pass3.stop = stop1
    pass3.paym = null
    pass3.state = wait
    pass4.quit = stop2
    pass4.stop = stop0
    pass4.paym = null
    pass4.state = wait
-- specification  G (((((passStates[0] = wait ->  F passStates[0] = home) & (pas
sStates[1] = wait ->  F passStates[1] = home)) & (passStates[2] = wait ->  F pas
sStates[2] = home)) & (passStates[3] = wait ->  F passStates[3] = home)) & (pass
States[4] = wait ->  F passStates[4] = home))  is true
```

Рис. 2.7.

2.3.5. Разрешение второй спецификации. Параллельный режим.

1) Ввод третьего терминала. Вторая спецификация выполняется.

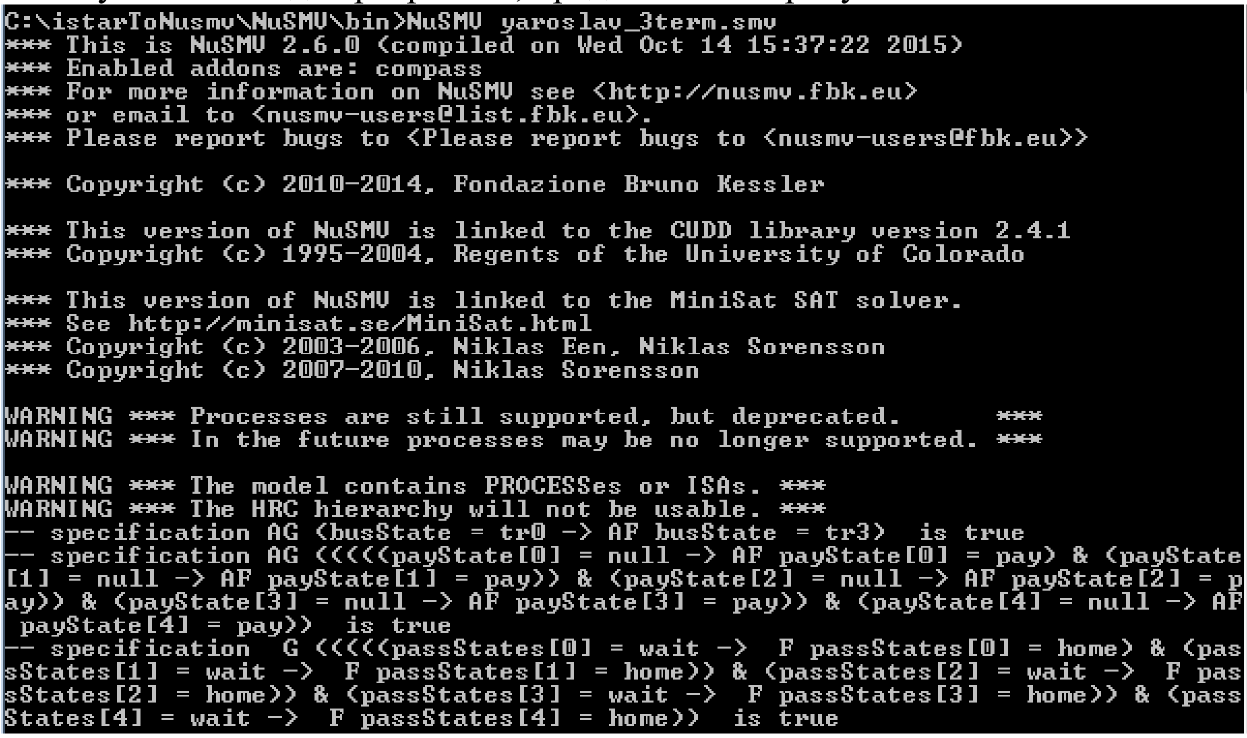
```
MODULE main
VAR
        pass0 : process passenger (0, passStates, doorState, doorOp, busState, terminal, payState, passStop, passQ, metk);
   pass1 : process passenger (1, passStates, doorState, doorOp, busState, terminal, payState, passStop, passQ, metk);
   pass2 : process passenger (2, passStates, doorState, doorOp, busState, terminal, payState, passStop, passQ, metk);
   pass3 : process passenger (3, passStates, doorState, doorOp, busState, terminal, payState, passStop, passQ, metk);
   pass4 : process passenger (4, passStates, doorState, doorOp, busState, terminal, payState, passStop, passQ, metk);
   passStates : array 0..4 of {wait, inn, payWait, payment, drive, outt, home};
   passStop : array 0..4 of {stop0, stop1};
   passQ : array 0..4 of {stop1, stop2};
   doorState : {allBusy, oneFree, twoFree, allFree};
   doorOp : {closed, opened};
   busState : {tr0, stop0, tr1, stop1, tr2, stop2, tr3};
   terminal : {allBusy, oneFree, allFree, allBusy2};
   payState : array 0..4 of {null, pay, not};
   metk : array 0..2 of {-1,0,1,2,3,4,5};
ASSIGN
   init(doorState) := allBusy;
   init(terminal) := allFree;
   init(busState) := tr0;
   init(doorOp) := closed;
   init(passStop[0]) := stop0;
   init(passStop[1]) := stop1;
   init(passStop[2]) := stop0;
   init(passStop[3]) := stop1;
   init(passStop[4]) := stop0;
   init(passQ[0]) := stop1;
   init(passQ[1]) := stop2;
   init(passQ[2]) := stop1;
   init(passQ[3]) := stop2;
   init(passQ[4]) := stop2;
   init(metk[0]) := 3;
   init(metk[1]) := 4;
   init(metk[2]) := 3;
-- Автобус проедет через все остановки
   CTLSPEC AG (busState = tr0 -> AF (busState = tr3))
-- Все кто зашел - оплатил проезд
   CTLSPEC AG ((payState[0] = null -> AF (payState[0] = pay)) & (payState[1] = null -> AF (payState[1] = pay)) & (payState[2] = null -> AF (payState[2] = pay)) & (payState[3] = null -> AF (payState[3] = pay)) & (payState[4] = null -> AF (payState[4]
```

```
= pay)))
-- Все окажутся дома
   LTLSPEC G ((passStates[0] = wait -> F passStates[0] = home) & (passStates[1] = wait
-> F passStates[1] = home) & (passStates[2] = wait -> F passStates[2] = home) &
(passStates[3] = wait -> F passStates[3] = home) & (passStates[4] = wait -> F
passStates[4] = home))
--Все, кто заплатил - уйдут
   LTLSPEC G ((payState[0] = pay -> F (passStates[0] = home)) & (payState[1] = pay ->
F (passStates[1] = home)) & (payState[2] = pay -> F (passStates[2] = home)) &
(payState[3] = pay -> F (passStates[3] = home)) & (payState[4] = pay -> F (passStates[4]
= home)))
MODULE passenger(id, passStates, doorState, doorOp, busState, terminal, payState,
passStop, passQ, metk)
VAR
   razr : {tr, tr1, fls};

DEFINE

        state := passStates[id];
   paym := payState[id];
   stop := passStop[id];
   quit := passQ[id];

ASSIGN

        init (passStates[id]) := wait;
   init (payState[id]) := null;
   init (razr) := tr;
next(passStop[id]) :=
   case
      TRUE : passStop[id];
   esac;
next(passQ[id]) :=
   case
      TRUE : quit;
   esac;
next(doorState) :=
   case
      state = outt & doorState = allFree & doorOp = opened : twoFree;
      state = outt & doorState = twoFree & doorOp = opened : oneFree;
      state = outt & doorState = oneFree & doorOp = opened : allBusy;
      state = inn & doorState = allFree & doorOp = opened : twoFree;
      state = inn & doorState = twoFree & doorOp = opened : oneFree;
      state = inn & doorState = oneFree & doorOp = opened : allBusy;
      state = home & doorOp = opened & doorState = allBusy : oneFree;
      state = home & doorOp = opened & doorState = oneFree : twoFree;
      state = home & doorOp = opened & doorState = twoFree : allFree;
      state = drive & doorOp = opened & doorState = allBusy : oneFree;
```

```
      state = drive & doorOp = opened & doorState = oneFree : twoFree;
      state = drive & doorOp = opened & doorState = twoFree : allFree;
      TRUE : doorState;
   esac;
next(doorOp) :=
   case
      doorOp = closed & busState = stop0 : opened;
      doorOp = closed & busState = stop1 : opened;
      doorOp = closed & busState = stop2 : opened;
      doorOp = opened & doorState = allFree & busState = stop0 & metk[0] = 0 : closed;
      doorOp = opened & doorState = allFree & busState = stop1 & metk[1] = 0 : closed;
      doorOp = opened & doorState = allFree & busState = stop2 & metk[2] = 0 : closed;
      TRUE : doorOp;
   esac;
next(metk[0]) :=
   case
      busState = stop0 & stop = stop0 & state = drive & metk[0] = 5 & razr = tr : 4;
      busState = stop0 & stop = stop0 & state = drive & metk[0] = 4 & razr = tr : 3;
      busState = stop0 & stop = stop0 & state = drive & metk[0] = 3 & razr = tr : 2;
      busState = stop0 & stop = stop0 & state = drive & metk[0] = 2 & razr = tr : 1;
      busState = stop0 & stop = stop0 & state = drive & metk[0] = 1 & razr = tr : 0;
      TRUE : metk[0];
   esac;
next(razr) :=
   case
      state = drive : tr1;
      state = home : fls;
      TRUE : razr;
   esac;
next(metk[1]) :=
   case
      busState = stop1 & stop = stop1 & state = drive & metk[1] = 5 & razr = tr : 4;
      busState = stop1 & stop = stop1 & state = drive & metk[1] = 4 & razr = tr : 3;
      busState = stop1 & stop = stop1 & state = drive & metk[1] = 3 & razr = tr : 2;
      busState = stop1 & stop = stop1 & state = drive & metk[1] = 2 & razr = tr : 1;
      busState = stop1 & stop = stop1 & state = drive & metk[1] = 1 & razr = tr : 0;
      busState = stop1 & quit = stop1 & state = home & metk[1] = 5 & razr = tr1 : 4;
      busState = stop1 & quit = stop1 & state = home & metk[1] = 4 & razr = tr1 : 3;
      busState = stop1 & quit = stop1 & state = home & metk[1] = 3 & razr = tr1 : 2;
      busState = stop1 & quit = stop1 & state = home & metk[1] = 2 & razr = tr1 : 1;
      busState = stop1 & quit = stop1 & state = home & metk[1] = 1 & razr = tr1 : 0;
      TRUE : metk[1];
   esac;
next(metk[2]) :=
   case
      busState = stop2 & quit = stop2 & state = home & metk[2] = 5 & razr = tr1 : 4;
      busState = stop2 & quit = stop2 & state = home & metk[2] = 4 & razr = tr1 : 3;
      busState = stop2 & quit = stop2 & state = home & metk[2] = 3 & razr = tr1 : 2;
```

```
      busState = stop2 & quit = stop2 & state = home & metk[2] = 2 & razr = tr1 : 1;
      busState = stop2 & quit = stop2 & state = home & metk[2] = 1 & razr = tr1 : 0;
      TRUE : metk[2];
   esac;
next(busState) :=
   case
      busState = tr0 : stop0;
      busState = stop0 & metk[0] = 0 & doorOp = closed : tr1;
      busState = tr1 : stop1;
      busState = stop1 & metk[1] = 0 & doorOp = closed : tr2;
      busState = tr2 : stop2;
      busState = stop2 & metk[2] = 0 & doorOp = closed : tr3;
      TRUE : busState;
   esac;
next(terminal) :=
   case
      state = payWait & terminal = allFree : oneFree;
      state = payWait & terminal = oneFree : allBusy;
      state = payWait & terminal = allBusy : allBusy2;
      state = drive & terminal = allBusy2 : allBusy;
      state = drive & terminal = allBusy : oneFree;
      state = drive & terminal = oneFree : allFree;
      TRUE : terminal;
   esac;
next(payState[id]) :=
   case
      state = payment & paym = null : pay;
      state = payWait & paym = null & terminal = allBusy2 : not;
      TRUE : paym;
   esac;
next(passStates[id]) :=
   case
      state = wait & stop = stop0 & busState = stop0 & doorOp = opened : inn;
      state = wait & stop = stop1 & busState = stop1 & doorOp = opened & doorState !=
allBusy : inn;
      state = inn : payWait;
      state = payWait & terminal != allBusy2 : payment;
      state = payment : drive;
      state = drive & quit = stop1 & busState = stop1 & doorOp = opened & doorState !=
allBusy : outt;
      state = drive & quit = stop2 & busState = stop2 & doorOp = opened & doorState !=
allBusy : outt;
      state = outt : home;
      --state = home : wait;
      TRUE : state;
   esac;
FAIRNESS
  running
```

Результат вывода программы, представлен на рисунке 2.8.



```
C:\istarToNusmv\NuSMV\bin>NuSMV yaroslav_3term.smv
*** This is NuSMV 2.6.0 (compiled on Wed Oct 14 15:37:22 2015)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <Please report bugs to <nusmv-users@fbk.eu>>

*** Copyright (c) 2010-2014, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson

WARNING *** Processes are still supported, but deprecated.      ***
WARNING *** In the future processes may be no longer supported. ***

WARNING *** The model contains PROCESSes or ISAs. ***
WARNING *** The HRC hierarchy will not be usable. ***
-- specification AG (busState = tr0 -> AF busState = tr3)  is true
-- specification AG ((((((payState[0] = null -> AF payState[0] = pay) & (payState
[1] = null -> AF payState[1] = pay)) & (payState[2] = null -> AF payState[2] = p
ay)) & (payState[3] = null -> AF payState[3] = pay)) & (payState[4] = null -> AF
 payState[4] = pay))  is true
-- specification  G (((((passStates[0] = wait ->  F passStates[0] = home) & (pas
sStates[1] = wait ->  F passStates[1] = home)) & (passStates[2] = wait ->  F pas
sStates[2] = home)) & (passStates[3] = wait ->  F passStates[3] = home)) & (pass
States[4] = wait ->  F passStates[4] = home))  is true
```

Рис. 2.8.

2) уменьшено количество пассажиров. Вторая спецификация выполняется.

```
MODULE main
VAR

        pass0 : process passenger (0, passStates, doorState, doorOp, busState, terminal,
payState, passStop, passQ, metk);
    pass1 : process passenger (1, passStates, doorState, doorOp, busState, terminal,
payState, passStop, passQ, metk);
    pass2 : process passenger (2, passStates, doorState, doorOp, busState, terminal,
payState, passStop, passQ, metk);
    pass3 : process passenger (3, passStates, doorState, doorOp, busState, terminal,
payState, passStop, passQ, metk);
    --pass4 : process passenger (4, passStates, doorState, doorOp, busState, terminal,
payState, passStop, passQ, metk);
    passStates : array 0..4 of {wait, inn, payWait, payment, drive, outt, home};
    passStop : array 0..4 of {stop0, stop1};
    passQ : array 0..4 of {stop1, stop2};
    doorState : {allBusy, oneFree, twoFree, allFree};
    doorOp : {closed, opened};
    busState : {tr0, stop0, tr1, stop1, tr2, stop2, tr3};
    terminal : {allBusy, oneFree, allFree, allBusy2};
    payState : array 0..4 of {null, pay, not};
    metk : array 0..2 of {-1,0,1,2,3,4,5};

ASSIGN
```

```
   init(doorState) := allBusy;
   init(terminal) := allFree;
   init(busState) := tr0;
   init(doorOp) := closed;
   init(passStop[0]) := stop0;
   init(passStop[1]) := stop1;
   init(passStop[2]) := stop0;
   init(passStop[3]) := stop1;
   init(passStop[4]) := stop0;
   init(passQ[0]) := stop1;
   init(passQ[1]) := stop2;
   init(passQ[2]) := stop1;
   init(passQ[3]) := stop2;
   init(passQ[4]) := stop2;
   init(metk[0]) := 2;
   init(metk[1]) := 4;
   init(metk[2]) := 2;
-- Автобус проедет через все остановки
   CTLSPEC AG (busState = tr0 -> AF (busState = tr3))
-- Все кто зашел - оплатил проезд
   CTLSPEC AG ((payState[0] = null -> AF (payState[0] = pay)) & (payState[1] = null -
> AF (payState[1] = pay)) & (payState[2] = null -> AF (payState[2] = pay)) &
(payState[3] = null -> AF (payState[3] = pay)))
-- Все окажутся дома
   LTLSPEC G ((passStates[0] = wait -> F passStates[0] = home) & (passStates[1] = wait
-> F passStates[1] = home) & (passStates[2] = wait -> F passStates[2] = home) &
(passStates[3] = wait -> F passStates[3] = home))
--Все, кто заплатил - уйдут
   LTLSPEC G ((payState[0] = pay -> F (passStates[0] = home)) & (payState[1] = pay ->
F (passStates[1] = home)) & (payState[2] = pay -> F (passStates[2] = home)) &
(payState[3] = pay -> F (passStates[3] = home)))
MODULE passenger(id, passStates, doorState, doorOp, busState, terminal, payState,
passStop, passQ, metk)
VAR
   razr : {tr, tr1, fls};
DEFINE
        state := passStates[id];
   paym := payState[id];
   stop := passStop[id];
   quit := passQ[id];
ASSIGN
        init (passStates[id]) := wait;
   init (payState[id]) := null;
   init (razr) := tr;
next(passStop[id]) :=
   case
      TRUE : passStop[id];
```

```
   esac;
next(passQ[id]) :=
   case
      TRUE : quit;
   esac;
next(doorState) :=
   case
      state = outt & doorState = allFree & doorOp = opened : twoFree;
      state = outt & doorState = twoFree & doorOp = opened : oneFree;
      state = outt & doorState = oneFree & doorOp = opened : allBusy;
      state = inn & doorState = allFree & doorOp = opened : twoFree;
      state = inn & doorState = twoFree & doorOp = opened : oneFree;
      state = inn & doorState = oneFree & doorOp = opened : allBusy;
      state = home & doorOp = opened & doorState = allBusy : oneFree;
      state = home & doorOp = opened & doorState = oneFree : twoFree;
      state = home & doorOp = opened & doorState = twoFree : allFree;
      state = drive & doorOp = opened & doorState = allBusy : oneFree;
      state = drive & doorOp = opened & doorState = oneFree : twoFree;
      state = drive & doorOp = opened & doorState = twoFree : allFree;
      TRUE : doorState;
   esac;
next(doorOp) :=
   case
      doorOp = closed & busState = stop0 : opened;
      doorOp = closed & busState = stop1 : opened;
      doorOp = closed & busState = stop2 : opened;
      doorOp = opened & doorState = allFree & busState = stop0 & metk[0] = 0 : closed;
      doorOp = opened & doorState = allFree & busState = stop1 & metk[1] = 0 : closed;
      doorOp = opened & doorState = allFree & busState = stop2 & metk[2] = 0 : closed;
      TRUE : doorOp;
   esac;
next(metk[0]) :=
   case
      busState = stop0 & stop = stop0 & state = drive & metk[0] = 5 & razr = tr : 4;
      busState = stop0 & stop = stop0 & state = drive & metk[0] = 4 & razr = tr : 3;
      busState = stop0 & stop = stop0 & state = drive & metk[0] = 3 & razr = tr : 2;
      busState = stop0 & stop = stop0 & state = drive & metk[0] = 2 & razr = tr : 1;
      busState = stop0 & stop = stop0 & state = drive & metk[0] = 1 & razr = tr : 0;
      TRUE : metk[0];
   esac;
next(razr) :=
   case
      state = drive : tr1;
      state = home : fls;
      TRUE : razr;
   esac;

next(metk[1]) :=
```

```
   case
      busState = stop1 & stop = stop1 & state = drive & metk[1] = 5 & razr = tr : 4;
      busState = stop1 & stop = stop1 & state = drive & metk[1] = 4 & razr = tr : 3;
      busState = stop1 & stop = stop1 & state = drive & metk[1] = 3 & razr = tr : 2;
      busState = stop1 & stop = stop1 & state = drive & metk[1] = 2 & razr = tr : 1;
      busState = stop1 & stop = stop1 & state = drive & metk[1] = 1 & razr = tr : 0;
      busState = stop1 & quit = stop1 & state = home & metk[1] = 5 & razr = tr1 : 4;
      busState = stop1 & quit = stop1 & state = home & metk[1] = 4 & razr = tr1 : 3;
      busState = stop1 & quit = stop1 & state = home & metk[1] = 3 & razr = tr1 : 2;
      busState = stop1 & quit = stop1 & state = home & metk[1] = 2 & razr = tr1 : 1;
      busState = stop1 & quit = stop1 & state = home & metk[1] = 1 & razr = tr1 : 0;
      TRUE : metk[1];
   esac;
next(metk[2]) :=
   case
      busState = stop2 & quit = stop2 & state = home & metk[2] = 5 & razr = tr1 : 4;
      busState = stop2 & quit = stop2 & state = home & metk[2] = 4 & razr = tr1 : 3;
      busState = stop2 & quit = stop2 & state = home & metk[2] = 3 & razr = tr1 : 2;
      busState = stop2 & quit = stop2 & state = home & metk[2] = 2 & razr = tr1 : 1;
      busState = stop2 & quit = stop2 & state = home & metk[2] = 1 & razr = tr1 : 0;
      TRUE : metk[2];
   esac;
next(busState) :=
   case
      busState = tr0 : stop0;
      busState = stop0 & metk[0] = 0 & doorOp = closed : tr1;
      busState = tr1 : stop1;
      busState = stop1 & metk[1] = 0 & doorOp = closed : tr2;
      busState = tr2 : stop2;
      busState = stop2 & metk[2] = 0 & doorOp = closed : tr3;
      TRUE : busState;
   esac;
next(terminal) :=
   case
      state = payWait & terminal = allFree : oneFree;
      state = payWait & terminal = oneFree : allBusy;
      --state = payWait & terminal = allBusy : allBusy2;
      --state = drive & terminal = allBusy2 : allBusy;
      state = drive & terminal = allBusy : oneFree;
      state = drive & terminal = oneFree : allFree;
      TRUE : terminal;
   esac;
next(payState[id]) :=
   case
      state = payment & paym = null : pay;
      state = payWait & paym = null & terminal = allBusy : not;
      TRUE : paym;
   esac;
```

```
next(passStates[id]) :=
   case
      state = wait & stop = stop0 & busState = stop0 & doorOp = opened : inn;
      state = wait & stop = stop1 & busState = stop1 & doorOp = opened & doorState !=
allBusy : inn;
      state = inn : payWait;
      state = payWait & terminal != allBusy : payment;
      state = payment : drive;
      state = drive & quit = stop1 & busState = stop1 & doorOp = opened & doorState !=
allBusy : outt;
      state = drive & quit = stop2 & busState = stop2 & doorOp = opened & doorState !=
allBusy : outt;
      state = outt : home;
      --state = home : wait;
      TRUE : state;
   esac;
FAIRNESS
  running
```

Результат вывода программы, представлен на рисунке 2.9.

```
C:\istarToNusmv\NuSMV\bin>NuSMV yaroslav_4pass.smv
*** This is NuSMV 2.6.0 (compiled on Wed Oct 14 15:37:22 2015)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <Please report bugs to <nusmv-users@fbk.eu>>

*** Copyright (c) 2010-2014, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson

WARNING *** Processes are still supported, but deprecated.      ***
WARNING *** In the future processes may be no longer supported. ***

WARNING *** The model contains PROCESSes or ISAs. ***
WARNING *** The HRC hierarchy will not be usable. ***
-- specification AG (busState = tr0 -> AF busState = tr3)  is true
-- specification AG ((((payState[0] = null -> AF payState[0] = pay) & (payState[
1] = null -> AF payState[1] = pay)) & (payState[2] = null -> AF payState[2] = pa
y)) & (payState[3] = null -> AF payState[3] = pay))  is true
-- specification  G ((((passStates[0] = wait ->  F passStates[0] = home) & (pass
States[1] = wait ->  F passStates[1] = home)) & (passStates[2] = wait ->  F pass
States[2] = home)) & (passStates[3] = wait ->  F passStates[3] = home))  is true

-- specification  G ((((payState[0] = pay ->  F passStates[0] = home) & (payStat
e[1] = pay ->  F passStates[1] = home)) & (payState[2] = pay ->  F passStates[2]
 = home)) & (payState[3] = pay ->  F passStates[3] = home))  is true
```

Рис. 2.9.

2.3.6. Нарушение второй спецификации. Параллельный режим. Введение счетчиков времени.

```
MODULE main
VAR
        pass0 : process passenger (0, passStates, doorState, doorOp, busState, terminal,
payState, passStop, passQ, metk);
   pass1 : process passenger (1, passStates, doorState, doorOp, busState, terminal,
payState, passStop, passQ, metk);
   pass2 : process passenger (2, passStates, doorState, doorOp, busState, terminal,
payState, passStop, passQ, metk);
   pass3 : process passenger (3, passStates, doorState, doorOp, busState, terminal,
payState, passStop, passQ, metk);
   pass4 : process passenger (4, passStates, doorState, doorOp, busState, terminal,
payState, passStop, passQ, metk);
   passStates : array 0..4 of {wait, inn, payWait, payment, drive, outt, home};
   passStop : array 0..4 of {stop0, stop1};
   passQ : array 0..4 of {stop1, stop2};
   doorState : {allBusy, oneFree, twoFree, allFree};
   doorOp : {closed, opened};
   busState : {tr0, stop0, tr1, stop1, tr2, stop2, tr3};
   terminal : {allBusy, oneFree, allFree, allBusy2};
   payState : array 0..4 of {null, pay, not};
   metk : array 0..2 of {-1,0,1,2,3,4,5};
ASSIGN
   init(doorState) := allBusy;
   init(terminal) := allFree;
   init(busState) := tr0;
   init(doorOp) := closed;
   init(passStop[0]) := stop0;
   init(passStop[1]) := stop1;
   init(passStop[2]) := stop0;
   init(passStop[3]) := stop1;
   init(passStop[4]) := stop0;
   init(passQ[0]) := stop1;
   init(passQ[1]) := stop2;
   init(passQ[2]) := stop1;
   init(passQ[3]) := stop2;
   init(passQ[4]) := stop2;
   init(metk[0]) := 3;
   init(metk[1]) := 4;
   init(metk[2]) := 3;
-- Автобус проедет через все остановки
   CTLSPEC AG (busState = tr0 -> AF (busState = tr3))
-- Все кто зашел - оплатил проезд
   CTLSPEC AG ((payState[0] = null -> AF (payState[0] = pay)) & (payState[1] = null -
> AF (payState[1] = pay)) & (payState[2] = null -> AF (payState[2] = pay)) &
(payState[3] = null -> AF (payState[3] = pay)) & (payState[4] = null -> AF (payState[4]
```

```
= pay)))
-- Все окажутся дома
   LTLSPEC G ((passStates[0] = wait -> F passStates[0] = home) & (passStates[1] = wait
-> F passStates[1] = home) & (passStates[2] = wait -> F passStates[2] = home) &
(passStates[3] = wait -> F passStates[3] = home) & (passStates[4] = wait -> F
passStates[4] = home))
--Все, кто заплатил - уйдут
   LTLSPEC G ((payState[0] = pay -> F (passStates[0] = home)) & (payState[1] = pay ->
F (passStates[1] = home)) & (payState[2] = pay -> F (passStates[2] = home)) &
(payState[3] = pay -> F (passStates[3] = home)) & (payState[4] = pay -> F (passStates[4]
= home)))
MODULE passenger(id, passStates, doorState, doorOp, busState, terminal, payState,
passStop, passQ, metk)
VAR
   razr : {tr, tr1, fls};
   time : 0..10;
   time_metk : {tr, fls};
 DEFINE
        state := passStates[id];
   paym := payState[id];
   stop := passStop[id];
   quit := passQ[id];
ASSIGN
        init (passStates[id]) := wait;
   init (payState[id]) := null;
   init (razr) := tr;
   init (time) := 0;
   init (time_metk) := tr;
next(passStop[id]) :=
   case
      TRUE : passStop[id];
   esac;
next(passQ[id]) :=
   case
      TRUE : quit;
   esac;
next(time) :=
   case
      state = payment & time_metk = tr & time !=10 : time + 1;
      TRUE : time;
   esac;
next(time_metk) :=
   case
      time_metk = tr & time = 10 : fls;
      TRUE : time_metk;
   esac;
next(doorState) :=
   case
```

```
      state = outt & doorState = allFree & doorOp = opened : twoFree;
      state = outt & doorState = twoFree & doorOp = opened : oneFree;
      state = outt & doorState = oneFree & doorOp = opened : allBusy;
      state = inn & doorState = allFree & doorOp = opened : twoFree;
      state = inn & doorState = twoFree & doorOp = opened : oneFree;
      state = inn & doorState = oneFree & doorOp = opened : allBusy;
      state = home & doorOp = opened & doorState = allBusy : oneFree;
      state = home & doorOp = opened & doorState = oneFree : twoFree;
      state = home & doorOp = opened & doorState = twoFree : allFree;
      state = drive & doorOp = opened & doorState = allBusy : oneFree;
      state = drive & doorOp = opened & doorState = oneFree : twoFree;
      state = drive & doorOp = opened & doorState = twoFree : allFree;
      TRUE : doorState;
   esac;
next(doorOp) :=
   case
      doorOp = closed & busState = stop0 : opened;
      doorOp = closed & busState = stop1 : opened;
      doorOp = closed & busState = stop2 : opened;
      doorOp = opened & doorState = allFree & busState = stop0 & metk[0] = 0 : closed;
      doorOp = opened & doorState = allFree & busState = stop1 & metk[1] = 0 : closed;
      doorOp = opened & doorState = allFree & busState = stop2 & metk[2] = 0 : closed;
      TRUE : doorOp;
   esac;
next(metk[0]) :=
   case
      busState = stop0 & stop = stop0 & state = drive & metk[0] = 5 & razr = tr : 4;
      busState = stop0 & stop = stop0 & state = drive & metk[0] = 4 & razr = tr : 3;
      busState = stop0 & stop = stop0 & state = drive & metk[0] = 3 & razr = tr : 2;
      busState = stop0 & stop = stop0 & state = drive & metk[0] = 2 & razr = tr : 1;
      busState = stop0 & stop = stop0 & state = drive & metk[0] = 1 & razr = tr : 0;
      TRUE : metk[0];
   esac;
next(razr) :=
   case
      state = drive : tr1;
      state = home : fls;
      TRUE : razr;
   esac;
next(metk[1]) :=
   case
      busState = stop1 & stop = stop1 & state = drive & metk[1] = 5 & razr = tr : 4;
      busState = stop1 & stop = stop1 & state = drive & metk[1] = 4 & razr = tr : 3;
      busState = stop1 & stop = stop1 & state = drive & metk[1] = 3 & razr = tr : 2;
      busState = stop1 & stop = stop1 & state = drive & metk[1] = 2 & razr = tr : 1;
      busState = stop1 & stop = stop1 & state = drive & metk[1] = 1 & razr = tr : 0;
      busState = stop1 & quit = stop1 & state = home & metk[1] = 5 & razr = tr1 : 4;
      busState = stop1 & quit = stop1 & state = home & metk[1] = 4 & razr = tr1 : 3;
```

```
      busState = stop1 & quit = stop1 & state = home & metk[1] = 3 & razr = tr1 : 2;
      busState = stop1 & quit = stop1 & state = home & metk[1] = 2 & razr = tr1 : 1;
      busState = stop1 & quit = stop1 & state = home & metk[1] = 1 & razr = tr1 : 0;
      TRUE : metk[1];
   esac;
next(metk[2]) :=
   case
      busState = stop2 & quit = stop2 & state = home & metk[2] = 5 & razr = tr1 : 4;
      busState = stop2 & quit = stop2 & state = home & metk[2] = 4 & razr = tr1 : 3;
      busState = stop2 & quit = stop2 & state = home & metk[2] = 3 & razr = tr1 : 2;
      busState = stop2 & quit = stop2 & state = home & metk[2] = 2 & razr = tr1 : 1;
      busState = stop2 & quit = stop2 & state = home & metk[2] = 1 & razr = tr1 : 0;
      TRUE : metk[2];
   esac;
next(busState) :=
   case
      busState = tr0 : stop0;
      busState = stop0 & metk[0] = 0 & doorOp = closed : tr1;
      busState = tr1 : stop1;
      busState = stop1 & metk[1] = 0 & doorOp = closed : tr2;
      busState = tr2 : stop2;
      busState = stop2 & metk[2] = 0 & doorOp = closed : tr3;
      TRUE : busState;
   esac;
next(terminal) :=
   case
      state = payWait & terminal = allFree : oneFree;
      state = payWait & terminal = oneFree : allBusy;
      --state = payWait & terminal = allBusy : allBusy2;
      --state = drive & terminal = allBusy2 : allBusy;
      state = drive & terminal = allBusy : oneFree;
      state = drive & terminal = oneFree : allFree;
      TRUE : terminal;
   esac;
next(payState[id]) :=
   case
      state = payment & paym = null : pay;
      state = payWait & paym = null & terminal = allBusy : not;
      TRUE : paym;
   esac;
next(passStates[id]) :=
   case
      state = wait & stop = stop0 & busState = stop0 & doorOp = opened : inn;
      state = wait & stop = stop1 & busState = stop1 & doorOp = opened & doorState !=
allBusy : inn;
      state = inn : payWait;
      state = payWait & payState[id] = not : drive;
      state = payWait & terminal != allBusy : payment;
```

```
        state = payment & time_metk = fls : drive;
        state = drive & quit = stop1 & busState = stop1 & doorOp = opened & doorState !=
allBusy : outt;
        state = drive & quit = stop2 & busState = stop2 & doorOp = opened & doorState !=
allBusy : outt;
        state = outt : home;
        --state = home : wait;
        TRUE : state;
    esac;
FAIRNESS
   running
```

Результат вывода программы, представлен на рисунке 2.10.

```
C:\istarToNusmv\NuSMV\bin>NuSMV yaroslavTimeS2.smv
*** This is NuSMV 2.6.0 (compiled on Wed Oct 14 15:37:22 2015)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <Please report bugs to <nusmv-users@fbk.eu>>

*** Copyright (c) 2010-2014, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson

WARNING *** Processes are still supported, but deprecated.      ***
WARNING *** In the future processes may be no longer supported. ***

WARNING *** The model contains PROCESSes or ISAs. ***
WARNING *** The HRC hierarchy will not be usable. ***
-- specification AG (busState = tr0 -> AF busState = tr3)  is true
-- specification AG (((((payState[0] = null -> AF payState[0] = pay) & (payState
[1] = null -> AF payState[1] = pay)) & (payState[2] = null -> AF payState[2] = p
ay)) & (payState[3] = null -> AF payState[3] = pay)) & (payState[4] = null -> AF
 payState[4] = pay))  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    pass0.razr = tr
    pass0.time = 0
    pass0.time_metk = tr
    pass1.razr = tr
    pass1.time = 0
    pass1.time_metk = tr
    pass2.razr = tr
    pass2.time = 0
    pass2.time_metk = tr
    pass3.razr = tr
    pass3.time = 0
    pass3.time_metk = tr
    pass4.razr = tr
    pass4.time = 0
    pass4.time_metk = tr
    passStates[0] = wait
    passStates[1] = wait
    passStates[2] = wait
    passStates[3] = wait
    passStates[4] = wait
    passStop[0] = stop0
    passStop[1] = stop1
    passStop[2] = stop0
    passStop[3] = stop1
    passStop[4] = stop0
    passQ[0] = stop1
    passQ[1] = stop2
    passQ[2] = stop1
    passQ[3] = stop2
    passQ[4] = stop2
    doorState = allBusy
    pass2.state = wait
    pass3.quit = stop2
    pass3.stop = stop1
    pass3.paym = null
    pass3.state = wait
    pass4.quit = stop2
    pass4.stop = stop0
    pass4.paym = null
    pass4.state = wait
-- specification  G (((((passStates[0] = wait ->  F passStates[0] = home) & (pas
sStates[1] = wait ->  F passStates[1] = home)) & (passStates[2] = wait ->  F pas
sStates[2] = home)) & (passStates[3] = wait ->  F passStates[3] = home)) & (pass
States[4] = wait ->  F passStates[4] = home))  is true
-- specification  G (((((payState[0] = pay ->  F passStates[0] = home) & (paySta
te[1] = pay ->  F passStates[1] = home)) & (payState[2] = pay ->  F passStates[2
] = home)) & (payState[3] = pay ->  F passStates[3] = home)) & (payState[4] = pa
y ->  F passStates[4] = home))  is true

C:\istarToNusmv\NuSMV\bin>
```

Рис. 2.10

2.3.7. Разрешение второй спецификации. Параллельный режим. Введение счетчиков времени.

1) Ввод третьего терминала. Вторая спецификация выполняется.

```
MODULE main
VAR
        pass0 : process passenger (0, passStates, doorState, doorOp, busState, terminal,
payState, passStop, passQ, metk);
   pass1 : process passenger (1, passStates, doorState, doorOp, busState, terminal,
payState, passStop, passQ, metk);
   pass2 : process passenger (2, passStates, doorState, doorOp, busState, terminal,
payState, passStop, passQ, metk);
   pass3 : process passenger (3, passStates, doorState, doorOp, busState, terminal,
payState, passStop, passQ, metk);
   pass4 : process passenger (4, passStates, doorState, doorOp, busState, terminal,
payState, passStop, passQ, metk);
   passStates : array 0..4 of {wait, inn, payWait, payment, drive, outt, home};
   passStop : array 0..4 of {stop0, stop1};
   passQ : array 0..4 of {stop1, stop2};
   doorState : {allBusy, oneFree, twoFree, allFree};
   doorOp : {closed, opened};
   busState : {tr0, stop0, tr1, stop1, tr2, stop2, tr3};
   terminal : {allBusy, oneFree, allFree, allBusy2};
   payState : array 0..4 of {null, pay, not};
   metk : array 0..2 of {-1,0,1,2,3,4,5};
ASSIGN
   init(doorState) := allBusy;
   init(terminal) := allFree;
   init(busState) := tr0;
   init(doorOp) := closed;
   init(passStop[0]) := stop0;
   init(passStop[1]) := stop1;
   init(passStop[2]) := stop0;
   init(passStop[3]) := stop1;
   init(passStop[4]) := stop0;
   init(passQ[0]) := stop1;
   init(passQ[1]) := stop2;
   init(passQ[2]) := stop1;
   init(passQ[3]) := stop2;
   init(passQ[4]) := stop2;
   init(metk[0]) := 3;
   init(metk[1]) := 4;
   init(metk[2]) := 3;
-- Автобус проедет через все остановки
   CTLSPEC AG (busState = tr0 -> AF (busState = tr3))
-- Все кто зашел - оплатил проезд
   CTLSPEC AG ((payState[0] = null -> AF (payState[0] = pay)) & (payState[1] =
null -> AF (payState[1] = pay)) & (payState[2] = null -> AF (payState[2] = pay)) &
```

```
(payState[3] = null -> AF (payState[3] = pay)) & (payState[4] = null -> AF
(payState[4] = pay)))
-- Все окажутся дома
   LTLSPEC G ((passStates[0] = wait -> F passStates[0] = home) & (passStates[1] =
wait -> F passStates[1] = home) & (passStates[2] = wait -> F passStates[2] = home)
& (passStates[3] = wait -> F passStates[3] = home) & (passStates[4] = wait -> F
passStates[4] = home))
--Все, кто заплатил - уйдут
   LTLSPEC G ((payState[0] = pay -> F (passStates[0] = home)) & (payState[1] =
pay -> F (passStates[1] = home)) & (payState[2] = pay -> F (passStates[2] = home))
& (payState[3] = pay -> F (passStates[3] = home)) & (payState[4] = pay -> F
(passStates[4] = home)))
MODULE passenger(id, passStates, doorState, doorOp, busState, terminal, payState,
passStop, passQ, metk)
VAR
   razr : {tr, tr1, fls};
   time : 0..10;
   time_metk : {tr, fls};
DEFINE
        state := passStates[id];
   paym := payState[id];
   stop := passStop[id];
   quit := passQ[id];
ASSIGN
        init (passStates[id]) := wait;
   init (payState[id]) := null;
   init (razr) := tr;
   init (time) := 0;
   init (time_metk) := tr;
next(passStop[id]) :=
   case
      TRUE : passStop[id];
   esac;
next(passQ[id]) :=
   case
      TRUE : quit;
   esac;
next(time) :=
   case
      state = payment & time_metk = tr & time !=10 : time + 1;
      TRUE : time;
   esac;
next(time_metk) :=
   case
      time_metk = tr & time = 10 : fls;
      TRUE : time_metk;
   esac;
next(doorState) :=
```

```
   case
      state = outt & doorState = allFree & doorOp = opened : twoFree;
      state = outt & doorState = twoFree & doorOp = opened : oneFree;
      state = outt & doorState = oneFree & doorOp = opened : allBusy;
      state = inn & doorState = allFree & doorOp = opened : twoFree;
      state = inn & doorState = twoFree & doorOp = opened : oneFree;
      state = inn & doorState = oneFree & doorOp = opened : allBusy;
      state = home & doorOp = opened & doorState = allBusy : oneFree;
      state = home & doorOp = opened & doorState = oneFree : twoFree;
      state = home & doorOp = opened & doorState = twoFree : allFree;
      state = drive & doorOp = opened & doorState = allBusy : oneFree;
      state = drive & doorOp = opened & doorState = oneFree : twoFree;
      state = drive & doorOp = opened & doorState = twoFree : allFree;
      TRUE : doorState;
   esac;
next(doorOp) :=
   case
      doorOp = closed & busState = stop0 : opened;
      doorOp = closed & busState = stop1 : opened;
      doorOp = closed & busState = stop2 : opened;
      doorOp = opened & doorState = allFree & busState = stop0 & metk[0] = 0 :
closed;
      doorOp = opened & doorState = allFree & busState = stop1 & metk[1] = 0 :
closed;
      doorOp = opened & doorState = allFree & busState = stop2 & metk[2] = 0 :
closed;
      TRUE : doorOp;
   esac;
next(metk[0]) :=
   case
      busState = stop0 & stop = stop0 & state = drive & metk[0] = 5 & razr = tr : 4;
      busState = stop0 & stop = stop0 & state = drive & metk[0] = 4 & razr = tr : 3;
      busState = stop0 & stop = stop0 & state = drive & metk[0] = 3 & razr = tr : 2;
      busState = stop0 & stop = stop0 & state = drive & metk[0] = 2 & razr = tr : 1;
      busState = stop0 & stop = stop0 & state = drive & metk[0] = 1 & razr = tr : 0;
      TRUE : metk[0];
   esac;
next(razr) :=
   case
      state = drive : tr1;
      state = home : fls;
      TRUE : razr;
   esac;
next(metk[1]) :=
   case
      busState = stop1 & stop = stop1 & state = drive & metk[1] = 5 & razr = tr : 4;
      busState = stop1 & stop = stop1 & state = drive & metk[1] = 4 & razr = tr : 3;
      busState = stop1 & stop = stop1 & state = drive & metk[1] = 3 & razr = tr : 2;
```

```
      busState = stop1 & stop = stop1 & state = drive & metk[1] = 2 & razr = tr : 1;
      busState = stop1 & stop = stop1 & state = drive & metk[1] = 1 & razr = tr : 0;
      busState = stop1 & quit = stop1 & state = home & metk[1] = 5 & razr = tr1 : 4;
      busState = stop1 & quit = stop1 & state = home & metk[1] = 4 & razr = tr1 : 3;
      busState = stop1 & quit = stop1 & state = home & metk[1] = 3 & razr = tr1 : 2;
      busState = stop1 & quit = stop1 & state = home & metk[1] = 2 & razr = tr1 : 1;
      busState = stop1 & quit = stop1 & state = home & metk[1] = 1 & razr = tr1 : 0;
      TRUE : metk[1];
   esac;
next(metk[2]) :=
   case
      busState = stop2 & quit = stop2 & state = home & metk[2] = 5 & razr = tr1 : 4;
      busState = stop2 & quit = stop2 & state = home & metk[2] = 4 & razr = tr1 : 3;
      busState = stop2 & quit = stop2 & state = home & metk[2] = 3 & razr = tr1 : 2;
      busState = stop2 & quit = stop2 & state = home & metk[2] = 2 & razr = tr1 : 1;
      busState = stop2 & quit = stop2 & state = home & metk[2] = 1 & razr = tr1 : 0;
      TRUE : metk[2];
   esac;
next(busState) :=
   case
      busState = tr0 : stop0;
      busState = stop0 & metk[0] = 0 & doorOp = closed : tr1;
      busState = tr1 : stop1;
      busState = stop1 & metk[1] = 0 & doorOp = closed : tr2;
      busState = tr2 : stop2;
      busState = stop2 & metk[2] = 0 & doorOp = closed : tr3;
      TRUE : busState;
   esac;
next(terminal) :=
   case
      state = payWait & terminal = allFree : oneFree;
      state = payWait & terminal = oneFree : allBusy;
      state = payWait & terminal = allBusy : allBusy2;
      state = drive & terminal = allBusy2 : allBusy;
      state = drive & terminal = allBusy : oneFree;
      state = drive & terminal = oneFree : allFree;
      TRUE : terminal;
   esac;
next(payState[id]) :=
   case
      state = payment & paym = null : pay;
      state = payWait & paym = null & terminal = allBusy2 : not;
      TRUE : paym;
   esac;
next(passStates[id]) :=
   case
      state = wait & stop = stop0 & busState = stop0 & doorOp = opened : inn;
      state = wait & stop = stop1 & busState = stop1 & doorOp = opened & doorState
```

```
!= allBusy : inn;
        state = inn : payWait;
        state = payWait & payState[id] = not : drive;
        state = payWait & terminal != allBusy2 : payment;
        state = payment & time_metk = fls : drive;
        state = drive & quit = stop1 & busState = stop1 & doorOp = opened & doorState
!= allBusy : outt;
        state = drive & quit = stop2 & busState = stop2 & doorOp = opened & doorState
!= allBusy : outt;
        state = outt : home;
        --state = home : wait;
        TRUE : state;
    esac;
FAIRNESS
   running
```

Результат вывода программы, представлен на рисунке 2.11.

```
C:\istarToNusmv\NuSMV\bin>NuSMV yaroslav_3term.smv
*** This is NuSMV 2.6.0 (compiled on Wed Oct 14 15:37:22 2015)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <Please report bugs to <nusmv-users@fbk.eu>>

*** Copyright (c) 2010-2014, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson

WARNING *** Processes are still supported, but deprecated.      ***
WARNING *** In the future processes may be no longer supported. ***

WARNING *** The model contains PROCESSes or ISAs. ***
WARNING *** The HRC hierarchy will not be usable. ***
-- specification AG (busState = tr0 -> AF busState = tr3)  is true
-- specification AG (((((payState[0] = null -> AF payState[0] = pay) & (payState
[1] = null -> AF payState[1] = pay)) & (payState[2] = null -> AF payState[2] = p
ay)) & (payState[3] = null -> AF payState[3] = pay)) & (payState[4] = null -> AF
 payState[4] = pay))  is true
-- specification  G (((((passStates[0] = wait ->  F passStates[0] = home) & (pas
sStates[1] = wait ->  F passStates[1] = home)) & (passStates[2] = wait ->  F pas
sStates[2] = home)) & (passStates[3] = wait ->  F passStates[3] = home)) & (pass
States[4] = wait ->  F passStates[4] = home))  is true
```

Рис. 2.11.

2) уменьшено количество пассажиров. Вторая спецификация выполняется.

```
MODULE main
VAR
        pass0 : process passenger (0, passStates, doorState, doorOp, busState, terminal,
payState, passStop, passQ, metk);
    pass1 : process passenger (1, passStates, doorState, doorOp, busState, terminal,
payState, passStop, passQ, metk);
    pass2 : process passenger (2, passStates, doorState, doorOp, busState, terminal,
payState, passStop, passQ, metk);
    pass3 : process passenger (3, passStates, doorState, doorOp, busState, terminal,
payState, passStop, passQ, metk);
```

```
    pass4 : process passenger (4, passStates, doorState, doorOp, busState, terminal, payState, passStop, passQ, metk);
    passStates : array 0..4 of {wait, inn, payWait, payment, drive, outt, home};
    passStop : array 0..4 of {stop0, stop1};
    passQ : array 0..4 of {stop1, stop2};
    doorState : {allBusy, oneFree, twoFree, allFree};
    doorOp : {closed, opened};
    busState : {tr0, stop0, tr1, stop1, tr2, stop2, tr3};
    terminal : {allBusy, oneFree, allFree, allBusy2};
    payState : array 0..4 of {null, pay, not};
    metk : array 0..2 of {-1,0,1,2,3,4,5};
ASSIGN
    init(doorState) := allBusy;
    init(terminal) := allFree;
    init(busState) := tr0;
    init(doorOp) := closed;
    init(passStop[0]) := stop0;
    init(passStop[1]) := stop1;
    init(passStop[2]) := stop0;
    init(passStop[3]) := stop1;
    init(passStop[4]) := stop0;
    init(passQ[0]) := stop1;
    init(passQ[1]) := stop2;
    init(passQ[2]) := stop1;
    init(passQ[3]) := stop2;
    init(passQ[4]) := stop2;
    init(metk[0]) := 3;
    init(metk[1]) := 4;
    init(metk[2]) := 3;
-- Автобус проедет через все остановки
    CTLSPEC AG (busState = tr0 -> AF (busState = tr3))
-- Все кто зашел - оплатил проезд
    CTLSPEC AG ((payState[0] = null -> AF (payState[0] = pay)) & (payState[1] = null -> AF (payState[1] = pay)) & (payState[2] = null -> AF (payState[2] = pay)) & (payState[3] = null -> AF (payState[3] = pay)) & (payState[4] = null -> AF (payState[4] = pay)))
-- Все окажутся дома
    LTLSPEC G ((passStates[0] = wait -> F passStates[0] = home) & (passStates[1] = wait -> F passStates[1] = home) & (passStates[2] = wait -> F passStates[2] = home) & (passStates[3] = wait -> F passStates[3] = home) & (passStates[4] = wait -> F passStates[4] = home))
--Все, кто заплатил - уйдут
    LTLSPEC G ((payState[0] = pay -> F (passStates[0] = home)) & (payState[1] = pay -> F (passStates[1] = home)) & (payState[2] = pay -> F (passStates[2] = home)) & (payState[3] = pay -> F (passStates[3] = home)) & (payState[4] = pay -> F (passStates[4] = home)))
MODULE passenger(id, passStates, doorState, doorOp, busState, terminal, payState, passStop, passQ, metk)
```

```
VAR
   razr : {tr, tr1, fls};
   time : 0..10;
   time_metk : {tr, fls};
 DEFINE
        state := passStates[id];
   paym := payState[id];
   stop := passStop[id];
   quit := passQ[id];
ASSIGN
        init (passStates[id]) := wait;
   init (payState[id]) := null;
   init (razr) := tr;
   init (time) := 0;
   init (time_metk) := tr;
next(passStop[id]) :=
   case
      TRUE : passStop[id];
   esac;
next(passQ[id]) :=
   case
      TRUE : quit;
   esac;
next(time) :=
   case
      state = payment & time_metk = tr & time !=10 : time + 1;
      TRUE : time;
   esac;
next(time_metk) :=
   case
      time_metk = tr & time = 10 : fls;
      TRUE : time_metk;
   esac;
next(doorState) :=
   case
      state = outt & doorState = allFree & doorOp = opened : twoFree;
      state = outt & doorState = twoFree & doorOp = opened : oneFree;
      state = outt & doorState = oneFree & doorOp = opened : allBusy;
      state = inn & doorState = allFree & doorOp = opened : twoFree;
      state = inn & doorState = twoFree & doorOp = opened : oneFree;
      state = inn & doorState = oneFree & doorOp = opened : allBusy;
      state = home & doorOp = opened & doorState = allBusy : oneFree;
      state = home & doorOp = opened & doorState = oneFree : twoFree;
      state = home & doorOp = opened & doorState = twoFree : allFree;
      state = drive & doorOp = opened & doorState = allBusy : oneFree;
      state = drive & doorOp = opened & doorState = oneFree : twoFree;
      state = drive & doorOp = opened & doorState = twoFree : allFree;
      TRUE : doorState;
```

```
    esac;
next(doorOp) :=
    case
        doorOp = closed & busState = stop0 : opened;
        doorOp = closed & busState = stop1 : opened;
        doorOp = closed & busState = stop2 : opened;
        doorOp = opened & doorState = allFree & busState = stop0 & metk[0] = 0 : closed;
        doorOp = opened & doorState = allFree & busState = stop1 & metk[1] = 0 : closed;
        doorOp = opened & doorState = allFree & busState = stop2 & metk[2] = 0 : closed;
        TRUE : doorOp;
    esac;
next(metk[0]) :=
    case
        busState = stop0 & stop = stop0 & state = drive & metk[0] = 5 & razr = tr : 4;
        busState = stop0 & stop = stop0 & state = drive & metk[0] = 4 & razr = tr : 3;
        busState = stop0 & stop = stop0 & state = drive & metk[0] = 3 & razr = tr : 2;
        busState = stop0 & stop = stop0 & state = drive & metk[0] = 2 & razr = tr : 1;
        busState = stop0 & stop = stop0 & state = drive & metk[0] = 1 & razr = tr : 0;
        TRUE : metk[0];
    esac;
next(razr) :=
    case
        state = drive : tr1;
        state = home : fls;
        TRUE : razr;
    esac;
next(metk[1]) :=
    case
        busState = stop1 & stop = stop1 & state = drive & metk[1] = 5 & razr = tr : 4;
        busState = stop1 & stop = stop1 & state = drive & metk[1] = 4 & razr = tr : 3;
        busState = stop1 & stop = stop1 & state = drive & metk[1] = 3 & razr = tr : 2;
        busState = stop1 & stop = stop1 & state = drive & metk[1] = 2 & razr = tr : 1;
        busState = stop1 & stop = stop1 & state = drive & metk[1] = 1 & razr = tr : 0;
        busState = stop1 & quit = stop1 & state = home & metk[1] = 5 & razr = tr1 : 4;
        busState = stop1 & quit = stop1 & state = home & metk[1] = 4 & razr = tr1 : 3;
        busState = stop1 & quit = stop1 & state = home & metk[1] = 3 & razr = tr1 : 2;
        busState = stop1 & quit = stop1 & state = home & metk[1] = 2 & razr = tr1 : 1;
        busState = stop1 & quit = stop1 & state = home & metk[1] = 1 & razr = tr1 : 0;
        TRUE : metk[1];
    esac;
next(metk[2]) :=
    case
        busState = stop2 & quit = stop2 & state = home & metk[2] = 5 & razr = tr1 : 4;
        busState = stop2 & quit = stop2 & state = home & metk[2] = 4 & razr = tr1 : 3;
        busState = stop2 & quit = stop2 & state = home & metk[2] = 3 & razr = tr1 : 2;
        busState = stop2 & quit = stop2 & state = home & metk[2] = 2 & razr = tr1 : 1;
        busState = stop2 & quit = stop2 & state = home & metk[2] = 1 & razr = tr1 : 0;
        TRUE : metk[2];
```

```
   esac;
next(busState) :=
   case
      busState = tr0 : stop0;
      busState = stop0 & metk[0] = 0 & doorOp = closed : tr1;
      busState = tr1 : stop1;
      busState = stop1 & metk[1] = 0 & doorOp = closed : tr2;
      busState = tr2 : stop2;
      busState = stop2 & metk[2] = 0 & doorOp = closed : tr3;
      TRUE : busState;
   esac;
next(terminal) :=
   case
      state = payWait & terminal = allFree : oneFree;
      state = payWait & terminal = oneFree : allBusy;
      state = payWait & terminal = allBusy : allBusy2;
      state = drive & terminal = allBusy2 : allBusy;
      state = drive & terminal = allBusy : oneFree;
      state = drive & terminal = oneFree : allFree;
      TRUE : terminal;
   esac;
next(payState[id]) :=
   case
      state = payment & paym = null : pay;
      state = payWait & paym = null & terminal = allBusy2 : not;
      TRUE : paym;
   esac;
next(passStates[id]) :=
   case
      state = wait & stop = stop0 & busState = stop0 & doorOp = opened : inn;
      state = wait & stop = stop1 & busState = stop1 & doorOp = opened & doorState !=
allBusy : inn;
      state = inn : payWait;
      state = payWait & payState[id] = not : drive;
      state = payWait & terminal != allBusy2 : payment;
      state = payment & time_metk = fls : drive;
      state = drive & quit = stop1 & busState = stop1 & doorOp = opened & doorState !=
allBusy : outt;
      state = drive & quit = stop2 & busState = stop2 & doorOp = opened & doorState !=
allBusy : outt;
      state = outt : home;
      --state = home : wait;
      TRUE : state;
   esac;
FAIRNESS
  running
```

Результат вывода программы, представлен на рисунке 2.12.

```
C:\istarToNusmv\NuSMV\bin>NuSMV yaroslav_4pass.smv
*** This is NuSMV 2.6.0 (compiled on Wed Oct 14 15:37:22 2015)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <Please report bugs to <nusmv-users@fbk.eu>>

*** Copyright (c) 2010-2014, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson

WARNING *** Processes are still supported, but deprecated.      ***
WARNING *** In the future processes may be no longer supported. ***

WARNING *** The model contains PROCESSes or ISAs. ***
WARNING *** The HRC hierarchy will not be usable. ***
-- specification AG (busState = tr0 -> AF busState = tr3)  is true
-- specification AG ((((payState[0] = null -> AF payState[0] = pay) & (payState[
1] = null -> AF payState[1] = pay)) & (payState[2] = null -> AF payState[2] = pa
y)) & (payState[3] = null -> AF payState[3] = pay))  is true
-- specification  G ((((passStates[0] = wait ->  F passStates[0] = home) & (pass
States[1] = wait ->  F passStates[1] = home)) & (passStates[2] = wait ->  F pass
States[2] = home)) & (passStates[3] = wait ->  F passStates[3] = home))  is true

-- specification  G ((((payState[0] = pay ->  F passStates[0] = home) & (payStat
e[1] = pay ->  F passStates[1] = home)) & (payState[2] = pay ->  F passStates[2]
 = home)) & (payState[3] = pay ->  F passStates[3] = home))  is true
```

Рис. 2.12.

2.4. ВЫВОД

NuSMV позволяет описывать модели и проверять необходимые свойства. Средство позволяет задавать как LTL, так и CTL спецификации.

NuSMV выдает подробное сообщение о состоянии системы в случае, если не проходит какое-то правило спецификации.

Все спецификации, описанные в данной работе, успешно прошли проверку.

III. Список использованных источников

1. Для тех, кому в IT-стартапе требуется разбор Си++ кода [Электронный ресурс], HABRAHABR. — URL: https://habr.com/company/intel/blog/99663/ (дата обращения: 2018-10-16).
2. В.М. Ицыксон. Методы обеспечения качества программных систем [Электронный ресурс], Институт компьютерных наук и технологий. — URL: http://kspt.icc.spbstu.ru/media/files/2016/course/softwarequality/QA2016_01_program_models.pdf (дата обращения: 2018-10-16).
3. И.В. Шошмина, Ю.Г. Карпов. Введение в язык Promela и систему комплексной верификации Spin
4. NuSMV Tutorial [Электронный ресурс]

URL: http://nusmv.fbk.eu/NuSMV