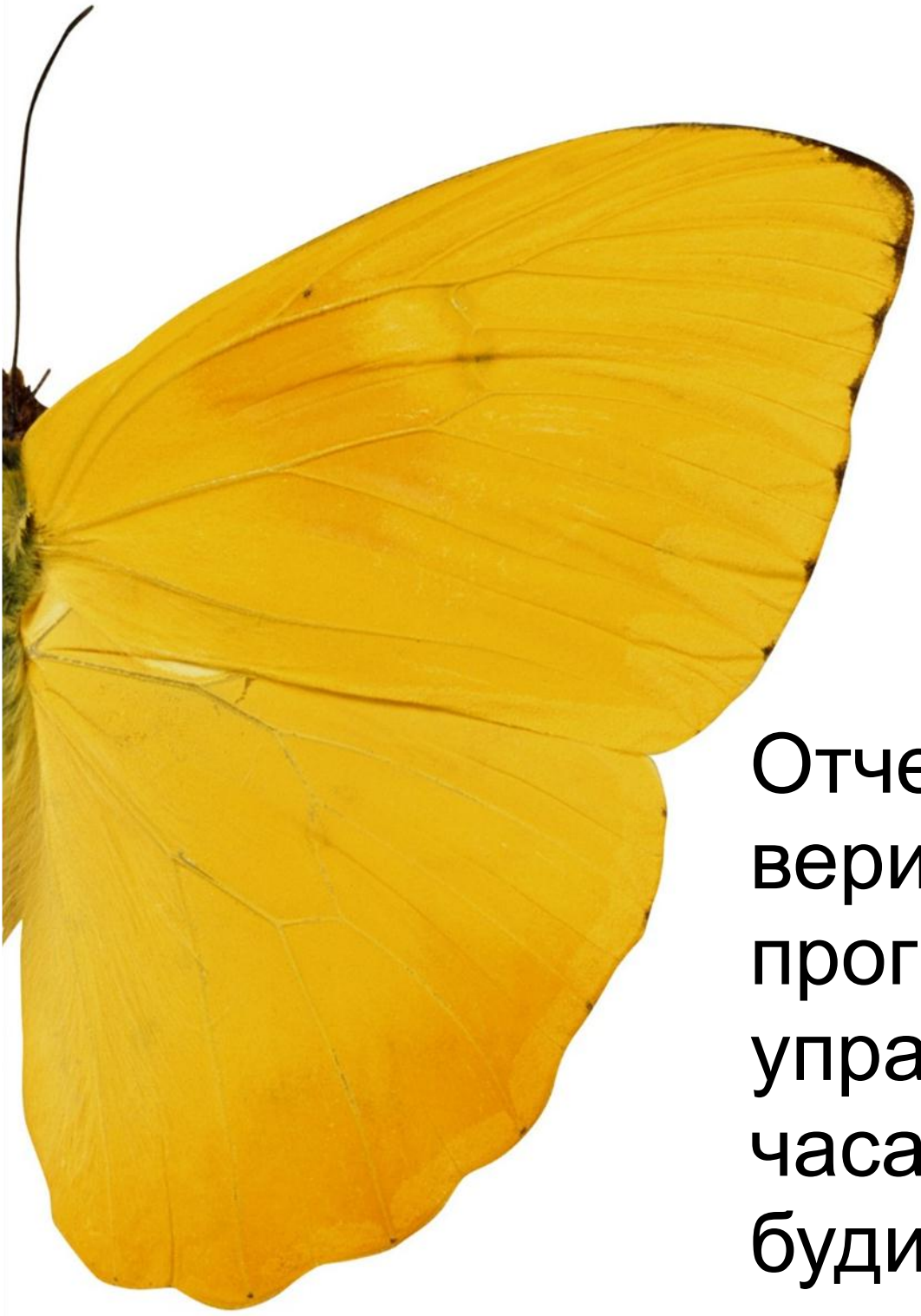

Отчет о верификации программы управления часами с будильником

Владимир Ульянцев, гр. 6538, НИУ ИТМО

ВВЕДЕНИЕ

В настоящем отчете приводится описание процесса верификации программы управления часами с будильником. Данная программа была составлена на основе управляющего автомата, приведенного в книге [1]. Верификация проводится с помощью программного средства *Spin* (http://spinroot.com).

ОПИСАНИЕ АВТОМАТА УПРАВЛЕНИЯ ЧАСАМИ

Автомат управления часами с будильником приведен на рис. 1. Стартовым состоянием автомата является состояние «Будильник выключен».

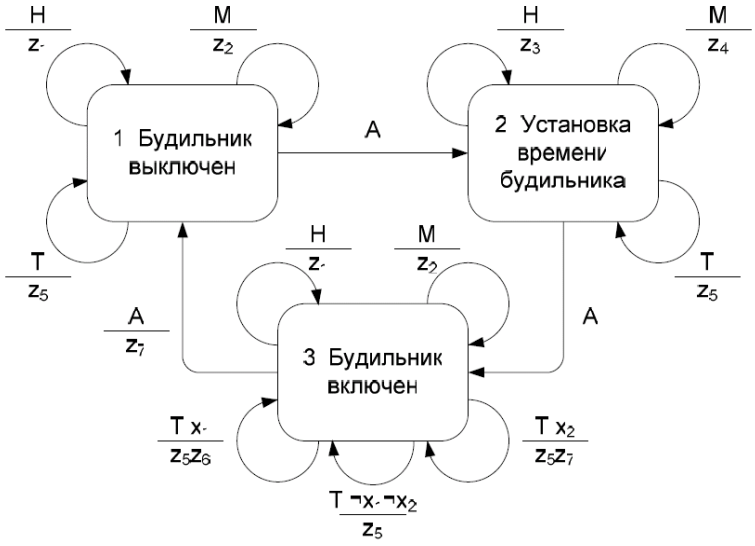

Рис. 1. Граф переходов конечного автомата управления часами с будильником

Опишем значения используемых событий:

- *H* – нажата кнопка «H» на корпусе часов;
- *M* – нажата кнопка «M» на корпусе часов;
- *A* – нажата кнопка «A» на корпусе часов;
- *T* – сработал таймер.

Используются две входные переменные:

- *x1* – верно ли, что время срабатывания будильника совпадает с текущим временем;
- *x2* – верно ли, что текущее время превышает время срабатывания будильника ровно на одну минуту.

Кроме этого эта автомат управления имеет семь выходных воздействий:

- *z1* – увеличить число часов текущего времени;
- *z2* – увеличить число минут часов текущего времени;
- *z3* – увеличить число часов времени срабатывания будильника;
- *z4* – увеличить число минут времени срабатывания будильника;
- *z5* – прибавить минуту к текущему времени;

- *z6* – включить звонок будильника;
- *z7* – выключить звонок будильника.

ПРОГРАММА НА ЯЗЫКЕ PROMELA

Представленный автомат был записан на языке *Promela*. Для сокращения числа состояний модели рассматривается время с 60 минутами в сутках. При использовании 1440 минут верификация невозможна из-за большого количества необходимой памяти.

```
// author: Vladimir Ulyantsev (ulyantsev@rain.ifmo.ru)

#define eA 1
#define eH 2
#define eM 3
#define eT 4
#define MOD 60

byte state = 0;
byte lastEvent;
bool isSoundOn = false;

int minutes = 0;
int alarmMinutes = 0;

bool x1;
bool x2;

inline z1() {
  minutes = (minutes + 60) % MOD;
}

inline z2() {
  minutes = (minutes + 1) % MOD;
}

inline z3() {
  alarmMinutes = (alarmMinutes + 60) % MOD;
}

inline z4() {
  alarmMinutes = (alarmMinutes + 1) % MOD;
}

inline calcX() {
  atomic {
    x1 = (minutes == alarmMinutes);
    x2 = (minutes == (alarmMinutes + 1) % MOD);
  }
}

inline ClockAutomata() {
  do
    ::(state == 0) ->
```

```
  printf("State 0: alarm is off\n");
  if
    ::lastEvent = eA;
      state = 1;
      printf("0 - A - 1\n");

    ::lastEvent = eM;
      z2();
      state = 0;
      printf("M button: time = %d\n", minutes);

    ::lastEvent = eH;
      z1();
      state = 0;
      printf("H button: time = %d\n", minutes);

    ::lastEvent = eT;
      z2();
      state = 0;
      printf("Tick: time = %d\n", minutes);

  fi;

::(state == 1) ->
  printf("State 1: set alarm time\n");
  if
    ::lastEvent = eA;
      state = 2;
      printf("1 - A - 2\n");

    ::lastEvent = eH;
      z3();
      state = 1;
      printf("H button: alarm time = %d\n", alarmMinutes);

    ::lastEvent = eM;
      z4();
      state = 1;
      printf("M button: alarm time = %d\n", alarmMinutes);

    ::lastEvent = eT;
      z2();
      state = 1;
      printf("Tick: time = %d\n", minutes);

  fi;

::(state == 2) ->
  printf("State 2: alarm is on\n");
  if
    ::lastEvent = eA;
      isSoundOn = false;
      state = 0;
      printf("2 - A / z7 - 0\n");
```

```
        ::lastEvent = eM;
          isSoundOn = false;
          z2();
          state = 2;
          printf("M button: time = %d\n", minutes);

        ::lastEvent = eH;
          isSoundOn = false;
          z1();
          state = 2;
          printf("H button: time = %d\n", minutes);

        ::lastEvent = eT;
          atomic {
            z2();
            state = 2;
            printf("Tick: time = %d\n", minutes);

            calcX();
            if
              ::(x1 == true) ->
                isSoundOn = true; //z6
                printf("Sound is on\n");
              ::(x2 == true) ->
                isSoundOn = false; //z7
                printf("Sound is off\n");
            fi;
          }
      fi;

  od;
}

proctype Model() {
  ClockAutomata();
}

init {
  run Model();
}
```

ИССЛЕДУЕМЫЕ ТЕМПОРАЛЬНЫЕ СВОЙСТВА

Был составлен набор темпоральных формул для проверки свойств приведенной программы.

1. `{ [] (isSoundOn -> (<>(!isSoundOn))) }` – данное свойство соответствует тому, что после того, как был включен звук будильника, он когда-нибудь будет выключен.

2. `{ [] (isSoundOn -> (state == 2 && x1)) }` – если звонок будильника включен, то автомат находится в состоянии «Будильник включен» и выполняется x_1.

3. `{[] (0 <= minutes && minutes < MOD && 0 <= alarmMinutes && alarmMinutes < MOD)}` – ограничения на допустимые значения переменных. В настоящей работе MOD равен 60.

4. `{[] (!(x1 && x2))}` – никогда одновременно не выполняется *x*1 и *x*2. Данное требование следует из определения данных переменных.

5. `{ [] (!(minutes == alarmMinutes) -> !isSoundOn) }` – если время часов и будильника не совпадают, то будильник не звонит.

ПРОВЕРКА ФОРМУЛ С ИСПОЛЬЗОВАНИЕМ SPIN

Проверка формулы производится при помощи следующей последовательности команд:

```
spin623.exe -a clock.pml
gcc pan.c -o pan.exe
pan.exe -a -m2000000 -I > pan_out.log
spin623.exe -k clock.pml.trail clock.pml > trail.log
```

Указанные темпоральные свойства были поочередно записаны в конце файла `clock.pml`. Каждая из них выполняется, если не вносить в программу изменений. Внесем ошибки в указанную программу и проверим составленные формулы.

Наиболее интересным случаем является проверка формулы 1. Оригинальный автомат, представленный на рис. 1 данной формуле не удовлетворяет. В самом деле, если будильник звонит, то можно нажать кнопку «M» два или более раз. Тем самым будильник будет звонить до тех пор, пока не будет нажата кнопка «A» или пока не произойдет выходное воздействие «z7». Таким образом, если закомментировать строку «`lastEvent = eM; //isSoundOn = false;`», будет выведен следующий контрпример:

```
ltl alarm_ltl: [] ((! (isSoundOn)) || (<> (! (isSoundOn))))
spin: couldn't find claim 2 (ignored)
              State 0: alarm is off
              0 - A - 1
              State 1: set alarm time
              M button: alarm time = 1
              State 1: set alarm time
              1 - A - 2
              State 2: alarm is on
              Tick: time = 1
              Sound is on
              State 2: alarm is on
              M button: time = 2
              State 2: alarm is on
              M button: time = 3
              State 2: alarm is on
              Tick: time = 4
spin: trail ends after 94 steps
```

```
#processes: 2
        state = 2
        lastEvent = 4
        isSoundOn = 1
        minutes = 4
        alarmMinutes = 1
        x1 = 0
        x2 = 0
 94:    proc  1 (Model) clock.pml:119 (state 77)
 94:    proc  0 (:init:) clock.pml:139 (state 2) <valid end state>
2 processes created
```

Также достаточно просто допустить ошибку, ведущую к невыполнению формулы 4. Достаточно забыть окружить atomic следующую часть кода, вычисляющую значения переменных:

```
  atomic {
    x1 = (minutes == alarmMinutes);
    x2 = (minutes == (alarmMinutes + 1) % MOD);
  }
```

ЗАКЛЮЧЕНИЕ

В настоящей работе рассмотрена программа управления часами с будильником на языке *Promela*. Для данной программы был проверен набор темпоральных формул при помощи программного средства *Spin*. Наиболее интересным результатом выполнения работы является нахождение ошибки в оригинальном автомате – часы с будильником могут звонить бесконечно долго.

ИСТОЧНИКИ

1. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. СПб: Питер, 2009.