SIMULATION AND VERIFICATION OF DIGITAL SYSTEMS WITH NUSMV

Daniela KOTMANOVÁ*
*Faculty of Informatics and Information Technologies,
Slovak University of Technology in Bratislava, Ilkovičova 3, 842 16 Bratislava 4, tel. 02/60 291 807,
e-mail: kotmanova@fiit.stuba.sk

ABSTRACT

We explore, in this work, some possibilities of the new model checker NuSMV, based on model checking algorithms. We started out from the former model checker SMV by verifying a very simple digital device, a two-state MicroWave Oven we designed, and then submitted the same device to the NuSMV Simulator and Model Checker. All results we obtained in verification and simulation with NuSMV are presented in the paper.

Keywords: *digital design, verification, temporal logic, model checking, SMV and NuSMV*

1. INTRODUCTION

The SMV model checker is one of the best known CTL (and LTL) model checkers. Recently, it was enlarged and improved (2007). Now called the checker NuSMV, it works on the same principles as SMV. We studied the new verification and simulation tool submitting the digital model to the simulation and model checking it provides.

1.1. MicroWave Oven design

The MicroWave Oven we present here is a simply electronic appliance for heating up meals. To set the device into action it is necessary to keep the door closed (and press the button Start).

We model the device as a Moore automaton (FSM).

st	cl
0	0
0	1
1	0
1	1

Next State	Idle	Heat
↓	Idle	Idle
	Idle	Idle
	Idle	Idle
	Heat	Heat

Current State	*y*
Idle	0
Defr	1

Fig. 1 State Transition and OutputTables

Transitions from:

- state Idle

Idle → Idle: $\neg st \vee \neg cl$
Idle → Heat: $st \wedge cl$

- state Heat

Heat → Heat: cl
Heat → Idle: $\neg st \vee \neg cl$

1.2. State Transition. Diagram (FSM)

The input to the device is given by the signals st and cl (close). State Idle is initial and the machine remains in state Idle until the signal st arrives on input. However, to move to the state Heat the signal cl must be true as well.

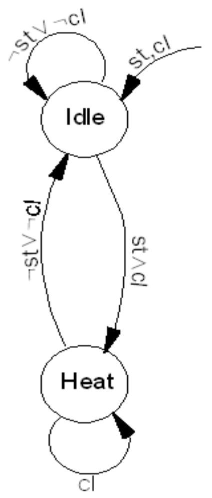


Fig. 2 State Transition Diagram

1.3. Temporal properties

The fundamental properties are:

```
AG(start&close -> AF status=Heat)     (1)

AG(~close ->AF ~status=Heat)          (2)
```

(1): If the start button is pressed and the door is closed the MicroWave Oven will always heat up.

(2): If the door is not closed the MicroWave Oven will never heat up (in all the states in the future).

Other properties are found in the programme 3.1.

We tried to apply also the operator Until U. As one can see, temporal properties with the temporal operator Until U do not hold for all states, and so are unusable for the verification (SMV program 3.1).

1.4. Kripke structure (State space model)

The state space model – representing a Kripke structure – is shown in Figure 2. The picture consists of eight global states, among them, four initial states. Global states are labelled with atomic propositions, each true in its state. Global transitions are drawn too, entirely.

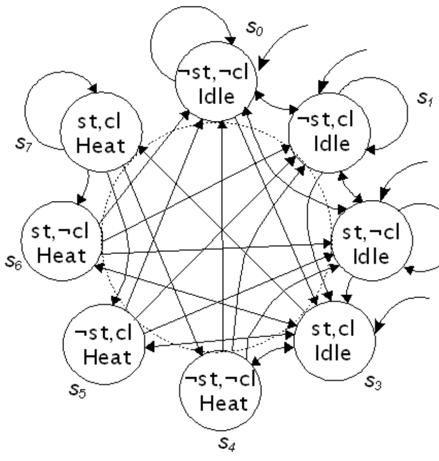


Fig. 3 State space model

The total number of global states in the state space model is given by 2^n where n is the number of state space variables.

The number of state space variables is determined by the number of input boolean variables augmented by the number of new boolean variables artificially created (the actual semantic states, ie., Idle and Heat, are coded with the 1-of-N code). So, we have now 4 state space boolean variables and 8 reachable states.

A global state is labeled with a proposition if and only if the proposition is true in it. Just true propositions about the defined semantic states are of some exception. If a real system is in one state it cannot be in an another. Thus, if a proposition, eg., *status*=Heat (or, in coded form, Heat =1), is true – what occurs in states s_4, s_5, s_6, s_7 in Fig. 3 – all other propositions concerning the other values of the variable *state* can be safely omitted, even true (in this case, the true proposition $\neg$*status*=Idle).

1.5. Parse Tree (Computer Tree)

Temporal formulas with CTL operators AG, EG, AF, EF,...etc., refer to execution paths. If we unwind the model in Fig. 3 into an infinite computation tree - parse tree (Fig. 4) we can visualize all execution paths from a given initial state. We chose among initial states the state s_0.

The validity of the propositions (1), (2) is explicitly visible on the tree. Each of them is valid in each state of the tree (operator AG). If the premises are false the implication is always true so we have to inspect only the global states where the premises are true to see if the conclusion is true as well. We set the colour to green for such states (for the 1st proposition). Since we did not want to over-colour the picture, for the true conclusions instead of the whole global state we coloured (to red) only the adequate proposition. For the truth value of the proposition (2), just the same.

We drew only four levels of the computation tree. From the 3th level on, the tree is incomplete, for not to overload the draft and keep it lucid. The tree goes on infinitely.

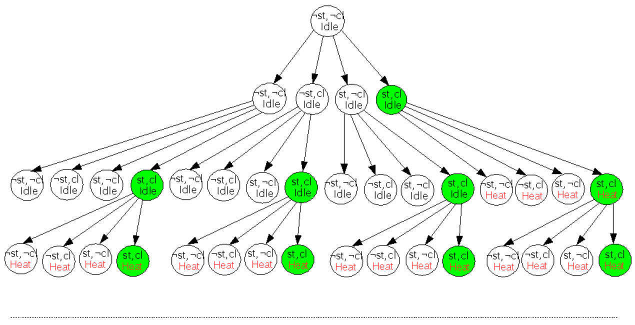


Fig. 4 Computation tree (infinite)

2. MODEL CHECKER SMV AND NUSMV

2.1. SMV Model Checker

SMV is an OBDD (Ordered Binary Decision Diagrams)-based model checker, developed by Ken McMillan at the Carnegie Mellon University (CMU), USA. It was the first symbolic model checker. (See in [3], [4], [5], [8].)

SMV Language and SMV Input File

SMV has its own language which includes also the syntax for LTL and CTL. The input to the SMV system is given by a program which includes the model and its specifications. The SMV produces as output the word „true" if the temporal specifications hold in the model, or shows a trace, ie. a sequence of global states, which indicates why the specification is false in the model.

SMV is available at the URL [8].

2.2. NuSMV Model Checker

NuSMV is the result of the joint project between CMU and ITC-IRST (Italy). NuSMV is a reimplementation and extension of SMV. NuSMV has both a batch and an interactive mode (interactive shell) and a draft of the NuSMV Graphical User Interface. We use only the interactive mode in this paper.

NuSMV is under ongoing development (included the GUI). We tried to examine the GUI under Windows XP but still without any worthy results.

NuSMV is available at the URL [9].

❖ NuSMV Checking specifications - Verification

NuSMV model checker works in the same way as the SMV checker: CTL or LTL specifications are evaluated by NuSMV in order to determine their truth value in the state space model. When a specification is found to be false, NuSMV produces a counterexample, ie. a trace in the state space model that falsifies the property.

NuSMV is an OBDD-based symbolic model checker, too.

❖ NuSMV Simulation

Moreover, NuSMV allows for the simulation of finite state models on the state-transition diagram (FSM) level. Hence, it may be used to check the adequacy of the FSM model.

NSMV simulation offers to the user the possibility of exploring all the possible executions of the model, before the verification of properties has taken place. The user tests the adequacy of the model by choosing input signals values to control the simulation.

2.3. Relation between Simulation and Verification

The relation between simulation and model checking is shown in Fig. 5. We have modified and completed the schema in [2] according to our own experiences with SMV and NuSMV. The schema represents steps in a Design Project Development.

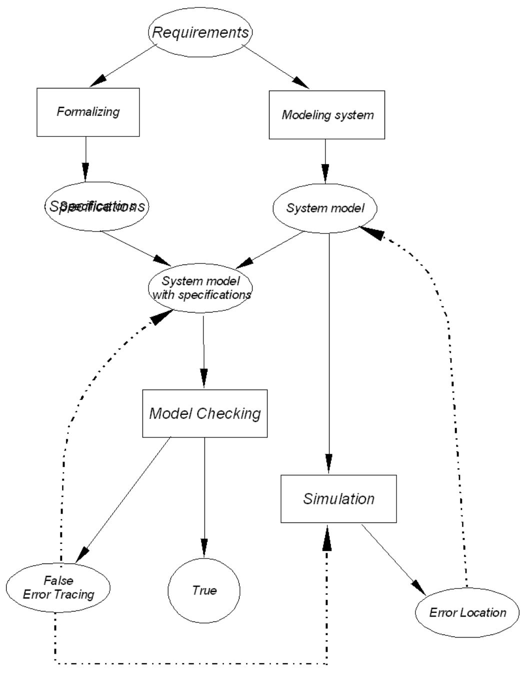


Fig. 5 Steps in the development of a digital design and relation between verification and simulation of a digital model

3. SMV MODEL CHECKING AND RESULTS

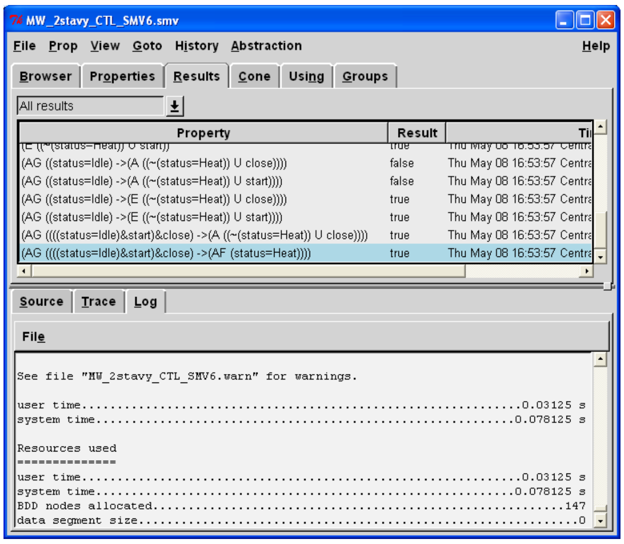
3.1. SMV program

```
MODULE main
VAR
   start  : boolean;
   close  : boolean;
      y   : boolean;

   status: {Idle,Heat};

ASSIGN
   init(status):=Idle;
/*Propositions E(~(status=Heat)U close)
and E(~(status=Heat) U start) true if
init state*/

   next(status):=case
          (start&close): Heat;
       1:Idle;
   esac;

       y:=(status=Heat);

/*Temporal specifications*/

SPEC AG(start&close -> AF status=Heat)
SPEC AG(~close ->AF ~status=Heat)
SPEC AG(~close ->AF status=Idle)

/*Unifying the last two into one*/
SPEC AG(~close ->AF~status=Heat|AF
status =Idle)

/*Remark:
Operator AF is right-distributive*/

SPEC AG((AF~status=Heat|AF status=Idle
)<->AF(~status=Heat|status=Idle))

/*Equivalence of the two states*/
SPEC AG(~status=Heat <-> status=Idle)

/*Exclusivity of global states*/
SPEC AG(status=Idle ^ status=Heat)

/*Output*/
SPEC AG(status=Idle <-> y=0)
SPEC AG(status=Heat <-> y=1)

/*Operator Until*/
-- Varieties.

SPEC AG((status=Idle)->EF close)
                               /*true*/
SPEC AG((status=Idle)->EF start)
                               /*true*/

SPEC A(~status=Heat U close)  /*false*/
SPEC A(~status=Heat U start)  /*false*/
SPEC E(~(status=Heat) U close)
/*true if init.state, false otherwise*/

SPEC E(~(status=Heat) U start)
/*true if init. state,false otherwise*/

SPECAG((status=Idle)->A(~(status=Heat )
U close))                     /*false*/
SPECAG((status=Idle)->A(~(status=Heat )
U start))                     /*false*/
```

```
SPEC AG((status=Idle)->E(~(status=Heat
) U close))                    /*true*/
SPEC AG((status=Idle) ->E(~(status=
Heat) U start))                /*true*/

/*Remark.*/
SPEC AG(((status=Idle)&(start&close))->
A(~(status=Heat)Uclose))       /*true*/
SPEC AG(((status=Idle)&(start&close))->
AF(status=Heat))               /*true*/
```

3.2. SMV Result Window

The SMV results can be seen in the SMV Result Window below (fragment).


MW_2stavy_CTL_SMV6.smv
File Prop View Goto History Abstraction Help
Browser Properties Results Cone Using Groups
All results
Source Trace Log
File

Property	Result	Ti
(E ((~(status=Heat)) U start))	true	Thu May 08 16:53:57 Centra
(AG ((status=Idle) ->(A ((~(status=Heat)) U close))))	false	Thu May 08 16:53:57 Centra
(AG ((status=Idle) ->(A ((~(status=Heat)) U start))))	false	Thu May 08 16:53:57 Centra
(AG ((status=Idle) ->(E ((~(status=Heat)) U close))))	true	Thu May 08 16:53:57 Centra
(AG ((status=Idle) ->(E ((~(status=Heat)) U start))))	true	Thu May 08 16:53:57 Centra
(AG ((((status=Idle)&start)&close) ->(A ((~(status=Heat)) U close))))	true	Thu May 08 16:53:57 Centra
(AG ((((status=Idle)&start)&close) ->(AF (status=Heat))))	true	Thu May 08 16:53:57 Centra

```
See file "MW_2stavy_CTL_SMV6.warn" for warnings.

user time..........................................................0.03125 s
system time.......................................................0.078125 s

Resources used
==============
user time..........................................................0.03125 s
system time.......................................................0.078125 s
BDD nodes allocated...................................................147
data segment size.......................................................0
```

Fig. 6 SMV Result Window

4. NUSMV INTERACTIVE ENVIRONMENT

The NuSMV interactive shell permits three different ways to generate a trace: deterministic, random and interactive. A trace is one of possible executions of the system.

Each of the three corresponds to a different way of how a starting state of the simulation is picked from a set of states and how the simulation progresses.

In deterministic simulation mode, as starting state the first state of the available set is chosen, whatever it is.

In the random mode, the starting state is chosen nondeterministically.

In the interactive simulation mode, the system stops at every steps, showing a list of possible future states and requesting the user to choose one of them. In this way, the user has a complete control over the progression of the simulation. Traces are now being formed by interactively adding the chosen states, one after one.

As for the other two simulation modes, the traces are generated by the NuSMV automatically and the user obtains the whole of the trace at a time and without seeing it at once - only after an appropriate command, with adequate parameters.

We offer a view of the use of each of them and a one interactive simulation step.

4.1. NuSMV Simulation

```
*** This is NuSMV 2.4.3 (compiled on
Tue May 22 14:08:54 UTC 2007)
*** For more information on NuSMV see
<http://nusmv.irst.itc.it> or email to
<nusmv-users@irst.itc.it>.
*** Please report bugs to
<nusmv@irst.itc.it>.

*** This version of NuSMV is linked to
the MiniSat SAT solver.
*** See
http://www.cs.chalmers.se/Cs/Research/F
ormalMethods/MiniSat
*** Copyright (c) 2003-2005, Niklas
Een, Niklas Sorensson

NuSMV>go                            <CR>
NuSMV>pick_state                    <CR>
NuSMV>print_current_state -v        <CR>

Current state is 1.1
start=0
close=0
y=0
status=Idle

NuSMV>pick_state -r                 <CR>
NuSMV>print_current_state -v        <CR>

Current state is 2.1
start=0
close=1
y=0
status=Idle

NuSMV>pick_state -i                 <CR>

*********  AVAILABLE STATES  *********
=============== State ================
0)start=1
  close=1
  y=0
  status=Idle
=============== State ================
1)start=0
=============== State ================
2)start=1
  close=0
=============== State ================
3)start=0

Choose a state from the above (0-3):0
                                   <CR>
Chosen state is: 0

NuSMV>print_current_state -v       <CR>

Current state is 3.1
start=1
close=1
y=0
status=Idle

NuSMV>show_traces                   <CR>

<!-- ###### Trace number: 3 ####### -->
```

```
Trace Description: Simulation Trace
Trace Type: Simulation
-> State: 3.1 <-
      start=1
      close=1
      y=0
      status=Idle

NuSMV>show_traces –t                <CR>

There are 3 traces currently available.

NuSMV>show_traces –v                <CR>

<!-- ###### Trace number: 3 ####### -->
Trace Description: Simulation Trace
Trace Type: Simulation
-> State: 3.1 <-
      start=1
      close=1
      y=0
      status=Idle

NuSMV>simulate 1                    <CR>

** Simulation Starting From State 3.1**

NuSMV>show_traces                   <CR>

<!-- ####### Trace number: 3 ###### -->
Trace Description: Simulation Trace
Trace Type: Simulation
-> State: 3.1 <-
      start=1
      close=1
      y=0
      status=Idle
-> Input: 3.2 <-
-> State: 3.2 <-
      start=0
      close=0
      y=1
      status=Heat

NuSMV>simulate -r 1                 <CR>

** Simulation Starting From State 3.2**

NuSMV>show_traces                   <CR>

<!-- ###### Trace number: 3 ####### -->
Trace Description: Simulation Trace
Trace Type: Simulation
-> State: 3.1 <-
      start=1
      close=1
      y=0
      status=Idle
-> Input: 3.2 <-
-> State: 3.2 <-
      start=0
      close=0
      y=1
      status=Heat
-> Input: 3.3 <-
-> State: 3.3 <-
      close=1
      y=0
      status=Idle
```

```
NuSMV>simulate -i 1                 <CR>

** Simulation Starting From State 3.3**

*********  AVAILABLE STATES  **********
=============== State =================
0)start=1
  close=1
  y=0
  status=Idle
=============== State =================
1)start=0
=============== State =================
2)start=1
  close=0
=============== State =================
3)start=0

Choose a state from the above (0-3): 0
                                    <CR>
Chosen state is: 0

NuSMV>show_traces

<!-- ###### Trace number: 3 ####### -->
Trace Description: Simulation Trace
Trace Type: Simulation
-- Loop starts here
-> State: 3.1 <-
      start=1
      close=1
      y=0
      status=Idle
-> Input: 3.2 <-
-> State: 3.2 <-
      start=0
      close=0
      y=1
      status=Heat
-> Input: 3.3 <-
-> State: 3.3 <-
      close=1
      y=0
      status=Idle
-> Input: 3.4 <-
-> State: 3.4 <-
      start=1

NuSMV>print_current_state           <CR>

Current state is 3.4

NuSMV>print_current_state –v        <CR>

Current state is 3.4
start=1
close=1
y=0
status=Idle

NuSMV> … …
```

Simulation constraints.

With the NuSMV simulator, we can exclude from the simulation the undesired states. Let us suppose we are not interested, for various reasons, in simulating state eg. start =0, close=0, status=Idle:

```
NuSMV>pick_state ………………-c “¬((status=
Idle)∧(¬start∧ ¬close))”          <CR>

NuSMV>simulate……………………-c “¬(((status=
Defrost)∧st2)∨((status=Heat)∧st1)∨(
st1∧st2))”                        <CR>
```

NuSMV simulator will take none of such states into account.

Remark.

If we had a MicroWave Oven with more states, eg, Defrost and Heat, the system could be simulated with manifested constraints. It this case, if each of states accepts only one input signal, say, state Defrost st1 and Heat st2, simulation constraint would be like this (excluding also the possibility of simultaneous arriving of st1 and st2 in any state):

```
NuSMV>pick_state………………-c “¬(((status=
Defrost)∧st2)∨((status=Heat)∧st1)∨(
st1∧st2))”                        <CR>

NuSMV>simulate……………………-c “¬(((status=
Defrost)∧st2)∨((status=Heat)∧st1)∨(
st1∧st2))”                        <CR>
```

4.2. NuSMV Model Checking

NuSMV allows not only to carry out a simulation of the model and check of properties but also other useful functionalities, such as checking the transition relation for totality (including computing and printing, if necessary, all reachable states), and some others.

```
NuSMV>go                          <CR>
NuSMV>compute_reachable           <CR>

Reachable States already enabled.

NuSMV>print_reachable_states      <CR>
#######################################
system diameter: 2
reachable states: 8(2^3) out of 16(2^4)
#######################################

NuSMV>print_reachable_states –v   <CR>

#######################################
system diameter: 2
reachable states: 8(2^3) out of 16(2^4)
------- State 1 ------
  start=1
  close=1
  y=0
  status=Idle
  status.0=1
------- State 2 ------
  start=0
  close=1
  y=0
  status=Idle
  status.0=1
------- State 3 ------
  start=1
  close=0
  y=0
  status=Idle
  status.0=1
------- State 4 ------
  start=0
  close=0
  y=0
  status=Idle
  status.0=1
------- State 5 ------
  start=1
  close=1
  y=1
  status=Heat
  status.0=0
------- State 6 ------
  start=0
  close=1
  y=1
  status=Heat
  status.0=0
------- State 7 ------
  start=1
  close=0
  y=1
  status=Heat
  status.0=0
------- State 8 ------
  start=0
  close=0
  y=1
  status=Heat
  status.0=0
#######################################

NuSMV>check_fsm                   <CR>

#######################################
The transition relation is total: No
deadlock state exists
#######################################

NuSMV>check_ctlspec -o ctl_spec.out
                                  <CR>

Output to file: ctl_spec.out

NuSMV> … … …
```

We can view the output file in Notepad++ in 4.3.

Model checking constraints.

With NuSMV model checker, we can choose a single property or some of properties to check:

```
NuSMV>check_ctlspec -p "AG((status=Idle
)->E[!(status=Heat)U close])"     <CR>

-- specification AG(status=Idle->E[!(
status=Heat)U close]) is true

NuSMV> …
```

4.3. NuSMV Output file

```
NuSMV>check_ctlspec -o ctl_spec.out                    <CR>
Output to file:ctl_spec.out
```

Output file ctl_spec.out open in Notepad++:

```
-- specification AG ((start & close) -> AF status = Heat) is true
-- specification AG (!close -> AF !(status = Heat))  is true
-- specification AG (!close -> AF status = Idle)  is true
-- specification AG (!close -> (AF !(status = Heat) | AF status = Idle))  is true
-- specification AG ((AF !(status = Heat) | AF status = Idle) <-> AF (!(status = Heat) | status = Idle))  is true
-- specification AG (!(status = Heat) <-> status = Idle)  is true
-- specification AG (status = Idle xor status = Heat)  is true
-- specification AG (status = Idle <-> y = 0)  is true
-- specification AG (status = Heat <-> y = 1)  is true
-- specification AG (status = Idle -> EF close)  is true
-- specification AG (status = Idle -> EF start)  is true
-- specification A [ !(status = Heat) U close ]   is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-- Loop starts here
-> State: 1.1 <-
  start = 0
  close = 0
  y = 0
  status = Idle
-> Input: 1.2 <-
-> State: 1.2 <-
-- specification A [ !(status = Heat) U start ]   is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-- Loop starts here
-> State: 2.1 <-
  start = 0
  close = 0
  y = 0
  status = Idle
-> Input: 2.2 <-
-> State: 2.2 <-
-- specification E [ !(status = Heat) U close ]   is true
-- specification E [ !(status = Heat) U start ]   is true
-- specification AG (status = Idle -> A [ !(status = Heat) U close ] )  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-- Loop starts here
-> State: 3.1 <-
  start = 0
  close = 0
  y = 0
  status = Idle
-> Input: 3.2 <-
-> State: 3.2 <-
-- specification AG (status = Idle -> A [ !(status = Heat) U start ] )  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-- Loop starts here
-> State: 4.1 <-
  start = 0
  close = 0
  y = 0
  status = Idle
-> Input: 4.2 <-
-> State: 4.2 <-
-- specification AG (status = Idle -> E [ !(status = Heat) U close ] )  is true
-- specification AG (status = Idle -> E [ !(status = Heat) U start ] )  is true
-- specification AG (((status = Idle & start) & close) -> A [ !(status = Heat) U close ] )  is true
-- specification AG (((status = Idle & start) & close) -> AF status = Heat)  is true
```

5. CONCLUSIONS

The aim of this paper was to make readers acquainted with some new tools of checking and verifying in Computer Science. In this context, we presented the outcomes we obtained by applying model checking verification method to the digital design, and using model checker SMV and its new version, the model checker (and simulator) NuSMV.

REFERENCES

[1] HUTH, M. - RYAN, M. D.: Logic in Computer Science. Modelling and Reasoning about Systems. Cambridge University Press, 2nd Ed. 2004 Cambridge, 427 pp.

[2] KATOEN, J. P.: Principles of Model Checking, Formal Methods and Tools Group, University of Twente. Lecture Notes 2004-5.

[3] McMiLLAN, K. L.: Cadence. Getting started with SMV. In: SMV Reference Manual. Cadence Berkeley Labs, Berkeley, 1999, USA.

[4] McMIILLAN, K. L: The Model Checking System. In: SMV Reference Manual. Cadence Berkeley Labs, Berkeley, 2002, USA

[5] McMILLAN, K. L. : Symbolic Model Checking. Kluwer Academic Publishers, 1993.

[6] CLARKE, E.M. - GRUMBERG, O. - LONG, D.E.: Verification Tools for Finite-State Concurrent Systems, LNCS 803. In: *The proceedings of REX school/symposium on A decade of concurrency: reflections and perspectives*, Noordwijkerhout, The Netherlands, June 1993.

[7] VARDI, Moshe Y.: Linear vs. Branching Time: A Complexity-Theoretic Perspective. In: *Proc. 13th IEEE Symposium on Logic in Computer Science*, pp. 394-405, 1998.

[8] SMV system draft. Available at: http://www.cs.cmu.edu/~modelcheck/smv.html

[9] NuSMV system draft, Tutorials, User Manuals Available at: http://nusmv.irst.itc.it/.

Received May 11, 2008, accepted August 21, 2009

BIOGRAPHY

Daniela Kotmanová graduated (MSc.) at the Faculty of Electrical Engineering and Informatics at Slovak University of Technology in Bratislava. She is a research worker at the Faculty of Informatics and Information Technologies, Slovak University of Technology, Bratislava. His works concerned domains of Temporal and Modal Logics, especially those related to Specification and Verification of Digital Systems, and to Models of Knowledge in Multi-agent Systems.