Temporal Specifications in Digital System Verification

Daniela Kotmanová*
daniela.kotmanova@gmail.com

Abstract

The thesis studies the applications of non-classical logics in the verification of digital systems. We focused mainly on finding how to establish temporal specifications describing digital model behaviour using a model checker SMV as a software verification tool. In the main part of our thesis we propose a method intended for use in digital sequential and combinational hardware circuit designs in order to verify them. Our method is presented in the form of a diagram with parts that are explained in detail. The method is illustrated through the examples of a digicode we have designed, specified and verified step by step according to the proposed method, and of simple Ready_Busy circuits. Further, we provide a formula to transform a digital model with an abstraction on the finite state machine level to the Kripke structure, which is, as we found out, identical in this case to the state space model.

In the experimental part of our thesis, we focus first on testing the method on several digital circuit designs and after that, on applying the proposed method on wholly new models. Concrete circuits have been designed, their dynamic behaviour has been described, temporal specifications have been expressed and finally, a verification program has been written.

The joint Appendix of our thesis contains all of the verification programs, result windows and output files.

In the Supplement, we compare reciprocally the character of non-classical logics and their relationships to the classical predicate logic.

Categories and Subject Descriptors

F.4.1 [**Mathematical Logic**]: Subject Temporal Logic; B.6.1 [**Design Styles**]: Subjects Combinational Logic, Sequential Circuits; B.6.3 [**Design Aids**]: Subject Verification

Keywords

temporal logic, model, specification, digital design, verification, model checking, model checker SMV

1. Introduction

Logics are an very important tool in the domain of Information and Communication System verification. This is true not only for the domain of reactive concurrent and transformation system models but also for the domain of hardware digital system models.

The state-of-the-art is, from the point of view of the verification, characterized by the predominance of classical techniques of checking correctness of the digital system design and implementation. These techniques are principally simulating and testing. However, neither of them can exhaustively check all of the behaviours of the model.

A new perspective is offered by the formal verification methods [11]: *deduction methods* and *model checking*.

Formal verification methods used in the development of information systems are mathematically-based techniques for describing (and verifying) system properties. They can be applied in the development of concurrent reactive systems as well as transformational systems (classical sequential programs) and also in the design of digital hardware systems.

Verification of a design in the true sense of the word means, in contrast with the simulation, an exhaustive check of the properties of the system being designed. The description of properties in the verification process is called a *specification*.

Verification of a system thus assumes, in the first place, the modeling of a system by using some of the modeling techniques, specifying the model by using some appropriate specification language and, in the end, the application of an adequate verification method and verification tool.

Formal verification methods are characterized:

Deduction methods. Verification by deduction methods uses a formal logic deductive apparatus consisting of the axioms and rules of inference, as referred to as a logic inference system. Unlike model checking, deduction methods are based on proving the validity of formulas in the

*Recommended by thesis supervisor: Assoc. Prof. Ladislav Hudec, PhD.
Defended at Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava on 2010.

given inference system Σ for temporal or even classical logic. The description of the system to verify is in a formal language and leads to a set Γ of formulas of the chosen logic. The specification φ is another formula of this logic. Verification then consists in finding, within the limits of the defined logic inference system Σ, a proof that φ logically follows from the set Γ:

$$\Gamma \vdash_{\Sigma} \varphi$$

The formal system Σ must be sound and complete.

Model checking. Verification by model checking is based on algorithms and uses computer verification software. The system to verify is represented by a finite model $\mathcal{M}$ for the language of some appropriate logic. To describe the system behaviour, i.e., to write specifications, formulas of this logic are used. Verification then consists in checking whether the model $\mathcal{M}$ satisfies the specification φ:

$$\mathcal{M} \vDash \varphi$$

The method verifies all of the states in the model, hence its name *"model checking"*.

2. Thesis Objectives

The thesis has several objectives:

- to propose and verify a method which allows quick and smart writing of temporal specifications for combinational and sequential digital circuits (Moore and Mealy Automata on infinite words) and to verify the model drawn, including the specifications, by using model checking as a verification method and model checker as a verification tool. The correct formulation of temporal properties is essential. Without a correct formulation of temporal properties, any verification activity is useless and unacceptable. In our verification procedure the establishment of the properties (writing of the formulas) of the model to be designed is done independently from the model design (construction of the Automaton) itself.
- to propose a formula for building Kripke structures starting from the finite-state machines (Moore or Mealy Automaton)
- to construct concrete Kripke structures, specifically for digital hardware sequential and combinational circuits, as the application field of the model checking method and the domain of the specification validity check.
- to find a syntactically correct structure for the output assignments for Moore and Mealy Automata in the language of the model checker SMV
- to activate and run new software verification tools, SMV and NuSMV, when NuSMV, the New Symbolic Model Verifier, still has some combined functionalities — for verification as well as for simulation.
- to better understand the semantics of the temporal logic language (as well as the modal one, intimately bound with) to make a comparison between the semantics of non-classical logics, such as temporal and modal, and to make an approach to the semantics of classical predicate logic language, pointing out the similarities and differences.

Our thesis is thus intended to redirect the attention of readers to a greater use of non classical logics in discrete system model verifications. This aim is also associated with the need to better implement new verification methods in practice.

The first acquisition of our thesis is the application of the model checking verification method to the finite-state machines such as Mealy and Moore Automata which behaviour we characterized by temporal formulas.

Moreover, the thesis will also stress the computer software support of verification methods and approach the possibilities of the exploitation of these new and powerful software tools.

3. Proposing a method to formulate and verify temporal specifications

At the moment, there are not precise instructions on how to proceed and correctly formulate temporal relations in the dynamics of a sequential hardware model (a Moore or Mealy Automaton) and how to correctly express relations between different variable quantities existing in the digital system model.

In our work, we take advantage, for that purpose, of the formalism of temporal logic.

Semantics of the language of temporal logic, given by the Kripke structure, i.e., by the triplet (S, ρ, L), have been applied to the design of digital systems, concretely to the finite state machines: Mealy and Moore Automata on infinite input word.

On the other hand, semantic of the temporal logic language is formally defined [11, 13] as a triplet (S, ρ, L), where

1. S is finite set of states
2. ρ is a binary relation on S, such that $\forall s \in S, s' \in S$ we have $s \rho s'$ (or $s \rightarrow s'$) and every $s \in S$ is reachable from the given initial state s_1: $\exists (s_1, s_2, \ldots, s_{n-1}, s) \in S^n : s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_{n-1} \rightarrow s$
3. L is a labeling function which associates to each state s from S a set of atomic propositions:

$$\begin{aligned} L : S &\rightarrow \wp(Atoms) \\ s &\mapsto L(s), L(s) \in \wp(Atoms) \end{aligned}$$

Atoms is a set of atomic propositions, $\wp(Atoms)$ is the power set of *Atoms*.

In our thesis, we have chosen:

1. as a set S the set of states in the state space built over the finite state machine (Automaton).
2. as a binary relation ρ the transition relation of the finite state machine, precisely, the transition relation of the state space model

3. as a labeling function L the function which assigns to each single state in the state space model a set of atomic propositions, possibly empty, which characterizes this state and is associated to it. Atomic propositions contain values of model variables.

In our thesis, we propose a method of establishing and verifying temporal formulas representing the behaviour of digital combinational and sequential models where the following conditions are assumed to hold throughout the thesis:

- the model has some initial state
- every state of the model has some successor state
- each state in the model is reachable from the initial state(s)

In our method, we present a digital model behaviour description which is independent of the character of the system being designed – it is not important if the modeled system is considered to be deterministic or non-deterministic. The proposed form of the behaviour description is valid for both.

The most important thing in the verification of discrete digital system designs - whether it is about verification by deduction methods or verification using model checking methods - is to find and construct an adequate model, which satisfies the task requirements, and to write appropriate temporal formulas (temporal properties), which express the required behaviour of the model.

To make the verification process we proposed more trustworthy we suggested to separate the formulation of the temporal properties from the design of the model. The principal idea is that for the verification of a model to be correct, the proceedings following the right- and left-hand branch of the Diagram in Figure 3 must be separated and independent.

That is to say, temporal specifications from the left-hand side of the Diagram in Figure 3 must be formed uniquely as temporal relations between user inputs and user outputs, starting from the requirements of the task, user inputs and outputs and initial situation being given by the requirements.

Digital model design from the right-hand side of the Diagram in Figure 3 is made by a particular and inventional approach. As a satisfying result of the abstract synthesis, including state reduction, a state-transition diagram of the finite state machine (FSM) representing the digital system model, with inputs and outputs, a defined set of states and state-transitions and some initial state, is fully accepted.

The input to the SMV model checker is given by a program written in the SMV language. In principle, the program consists of two main parts: the model description (design) and model specifications (behaviour).

Writing the input program simply means describing, by means of the model checker language, the model drawn and adding temporal specifications to the model. The model description will tell the computer what system it is going on and the specifications what behaviour the system model is presumed to accomplish.

The link between the two parts of a SMV program, i.e., between Temporal Specifications and Digital Model Design, will be realized in the input file to the model checker, in the part describing the model, by means of the program output variable quantities. The output variables are assigned some value of the symbolic state variable either separately (for Moore finite-state machine outputs) or as a logical product with the logical expression given by the combination of adequate user input-output values, that is to say, a disjunctive normal form (for Mealy finite-state machine outputs).

This way, taking into account the required type of user output in the time context relations (which finally amounts to a design of a Mealy or Moore finite-state machine) we found syntactically accepted constructs to give outputs in the SMV language:

Moore machine: the SMV output statement is a habitual assignment of the symbolic state variable value; if there are several outputs, the total output is the logical (exclusive) sum of the individual partial outputs associated with the relevant states.

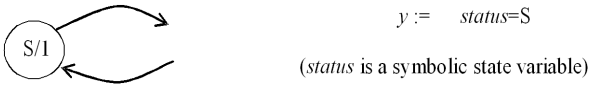

Figure 1: Moore Automaton - Output y in the SMV program

Mealy machine: the SMV output statement is a little more complicated. A disjunctive normal form is constructed from the input boolean variables associated to the current state and provoking (together with the state) an unitary output. Then a logical product of the disjunctive normal form and the symbolic state variable (precisely, with the appropriate value of the symbolic state variable) is made. This is a syntactic form for a partial output of a Mealy machine in the SMV language. The partial output is unitary if the disjunctive normal form takes on a value of logical 1 and simultaneously, the model is located in that state. We did a partial output in the SMV language for each state and its disjunctive normal form which is given by the unitary outputs of the system in this state. The total output in the SMV language is then a logical (exclusive) sum of the partial outputs (which are of the form of products). This is a correct and, in accord with the SMV language structure, syntactically accepted expression for the SMV output assignment statement.

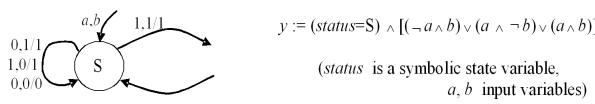

Figure 2: Mealy Automaton – Output y in the SMV program

Verification can be started from writing temporal formulas, i.e., following the left-hand side of the Diagram in

Figure 3, and then passing to the right-hand side to the abstract synthesis, or choosing the other way, starting by abstract synthesis and pursuing by creating temporal formulas. It does not matter what we choose first.

However, when constructing a Kripke structure we have to consider both at the same time; the temporal relations (left-hand side of the Diagram, Fig 3) and the semantic interpretation of temporal logic language, i.e., the states and transitions of the FSM being designed (right-hand side of the Diagram, Figure 3). On this basis we build a state space model using the conversion formula we found out. The model checker containing efficient algorithms will check the whole state space (the whole Kripke structure) and find out whether temporal specifications are valid for the given model or not.

Creating temporal specifications is initially not very self-evident. It is necessary to learn a style of reasoning which is called *"temporal thinking"*. However, with a certain practice, temporal specifications stem from themselves from the requirements of the task. Notwithstanding, in our method, to have a better view and orientation, we explain in details all of the steps essential for writing and verifying temporal specifications, together with an appropriate commentary. These detailed procedures need not necessarily be applied in the current life, inasmuch as they arise, as already stated, once model variables have been determined correctly, from the requirements themselves.

Procedure for writing temporal specifications and verifying digital design

The procedure as we have proposed it comprises several parts. In Figure 3, we illustrate the process through a Diagram.

4. Verification and Application of the method

We verified the proposed method with digital hardware systems:

- Digicode Device
- Ready-Busy Circuits
- Serial two-bit Adder
- Code Controller
- Binary Comparator
- Input Sequence Detectors
- Two-bit Arbiter

We applied the proposed method to our own model designs:

- Four-state MicroWave Oven with an Error State
- Simple MicroWave Oven with Two States
- Digicode with Error State

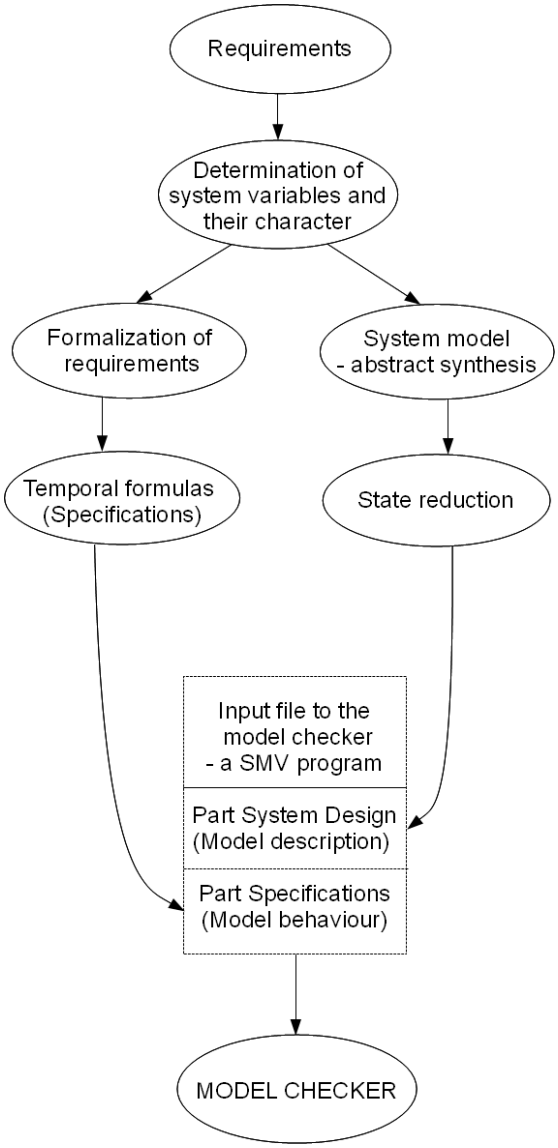

Figure 3: Procedure to form temporal formulas and verify a digital design.

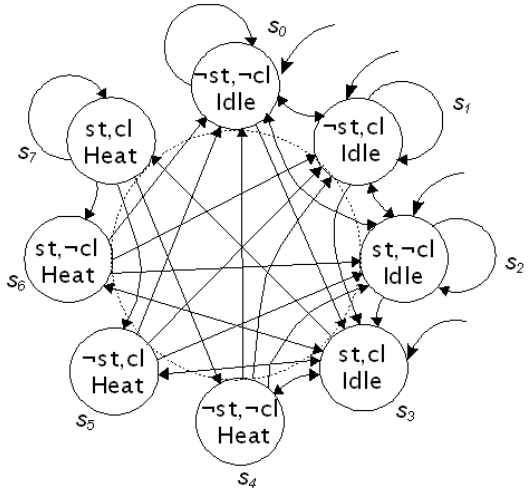

Figure 4: Kripke structure of a simple MicroWave Oven model with two states.

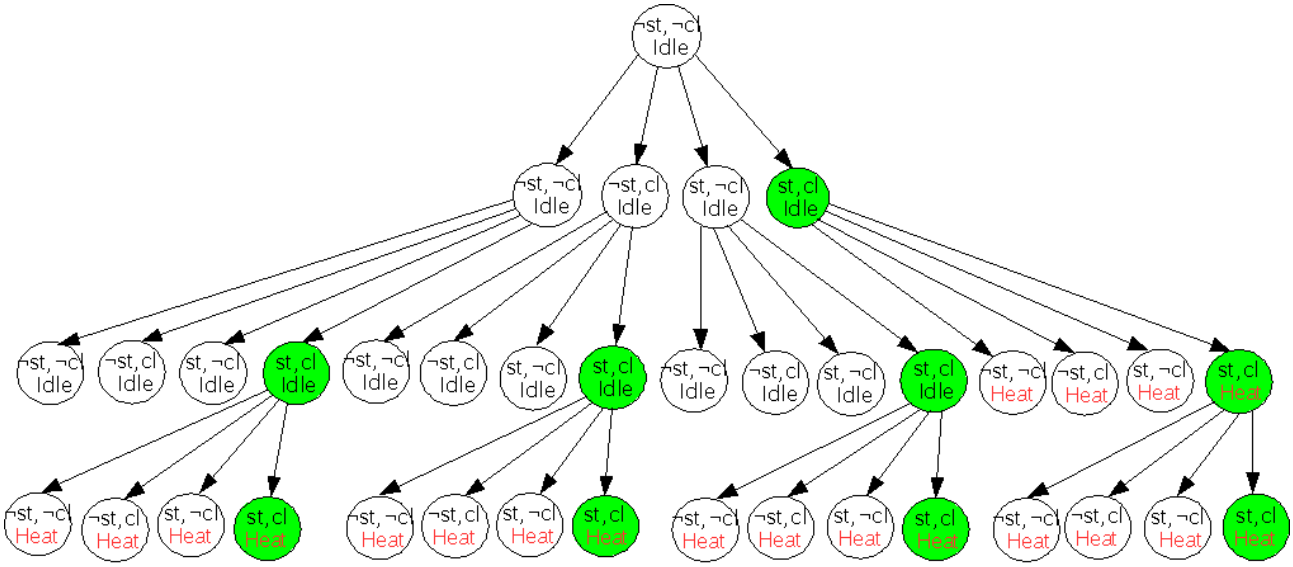

Figure 5: Unfolding the Kripke structure from Figure 4 into an infinite parse tree (starting from the initial state s_0; dashed line marking the infinite continuation of the parse tree).

5. Thesis Contributions

Main contributions can be resumed in the following points:

- We proposed a method to write and verify temporal specifications for digital hardware systems (combinational and sequential circuits) using model checking as verification method and the model checker SMV as computer software tool.
- We proposed a method to proceed in verifying digital designs: we drew a diagram, which illustrates step-by-step all of the procedures, from the task requirements through the design and specifications up to the verification. We made ourselves some examples as patterns - those of a Digicode and of a simple two-state Ready-Busy Circuit.
- The correctness of the verification in our method is guaranteed by the independence of the right- and left- hand sides of the Diagram, i.e., the independence of the hardware design model from the expressed temporal properties of the model, the latter given by the task requirements
- We found syntactic constructs about the SMV language output for the Moore and Mealy machine
- We applied the proposed method to new own designs we made (Four-state MicroWave Oven with Error State, Two-state simple MicroWave Oven, Digicode with Error State, example-patterns of a Simplified Digicode to illustrate the method, in the version for the Moore and Mealy finite state machines, and of a simple Ready-Busy Circuit)
- We provided a way to build a Kripke structure for digital sequential circuits to verify; for several of them, we also drew it concretely, including a combinational circuit. This has not been found in any literature
- We gave a general relation to convert a state-transition diagram as a model of a sequential digital system (Moore or Mealy Automaton on infinite words) to a Kripke structure (state space model)
- The resulting formula to transform a classical model of the sequential system to the Kripke structure has been verified by us with the new simulator and verificator NuSMV
- We came to the non-insignificant conclusion that in the case of digital sequential models, the Kripke structure is identical to the state space model; in reality model checking running on the Kripke structure runs on the state space model
- We succeed in animating and running new verification tools SMV and NuSMV, both downloaded free from the Internet network. Using these new tools, we could also by simulation separately verify the state–transition diagrams of digital models we designed
- We compared the semantics of the temporal logic language with the semantics of the modal logic language and both with the semantics of classical predicate logic and gave their relationships we arrived at

Acknowledgements. This work was supported by the Scientific Grant Agency of the Slovak Republic, Grant No. VG 2/5108/20 Graphic Simulation System for the Synchronization Logic Development and Debugging for Distributed Control of the Flexible Manufacturing Systems (1998-2000), Grant No. VG 2/1101/21 Real-time Models for Discrete Event Dynamic Systems and their Analyse (2001-2003), Grant No. VG 1/0157/03 Methods and Tools for Security and Management Development of Network and Mobile Computer Systems (2003 – 2005), Grant No. VG 1/3104/06 Grid Computing Systems and its Components (2006 – 2008).

References

[1] R. Alur, T. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM (JACM)*, 49(5):672–713, 2002.

[2] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, P. Schnoebelen, B. Berard, M. Bidoit, and A. Finkel. *Systems and software verification*. Springer, 2001.

[3] R. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys (CSUR)*, 24(3):293–318, 1992.

[4] E. Clarke, E. Emerson, and A. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.

[5] E. Clarke, O. Grumberg, and D. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(5):1512–1542, 1994.

[6] E. Clarke, O. Grumberg, and D. Long. Verification tools for finite-state concurrent systems. *A Decade of Concurrency Reflections and Perspectives*, pages 124–175, 1994.

[7] R. Cori and D. Lascar. *Logique mathématique: Calcul propostionnel, algèbres de Boole, calcul des prédicats*. Masson, 1993.

[8] O. Grumberg and D. Long. Model checking and modular verification. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(3):843–871, 1994.

[9] J. Halpern. Reasoning about knowledge: A survey. In *Handbook of logic in artificial intelligence and logic programming*. Citeseer, 1995.

[10] J. Halpern and M. Vardi. *Model checking vs. theorem proving: a manifesto*. Citeseer, 1991.

[11] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge Univ Pr, 2004.

[12] N. Jennings, P. Faratin, A. Lomuscio, S. Parsons, M. Wooldridge, and C. Sierra. Automated negotiation: prospects, methods and challenges. *Group Decision and Negotiation*, 10(2):199–215, 2001.

[13] J. Katoen. Principles of model-checking, formal methods and tools group. *University of Twente, Leeture Notes" System validation"(course 214012)*, 2003, 2002.

[14] J. Kolář, O. Štěpánková, and M. Chytil. *Logika, algebry a grafy*. SNTL, 1989.

[15] Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems: Specification*. springer, 1992.

[16] Z. Manna and A. Pnueli. *Temporal verification of reactive systems: safety*. Springer Verlag, 1995.

[17] K. McMillan. *Symbolic model checking*, volume 174. Kluwer Academic, 1993.

[18] K. McMillan. Getting started with SMV. *Cadence Berkeley Laboratories*, 1998.

[19] K. McMillan. The SMV system. *Cadence Berkeley Labs*, 1999.

[20] C. Meinel and T. Theobald. *Algorithms and data structures in VLSI design: OBDD-foundations and applications*. Springer Verlag, 1998.

[21] S. Russell and P. Norvig. *Artificial intelligence: a modern approach*. Prentice hall, 2009.

[22] V. Švejdar. *Logika: neúplnost, složitost a nutnost*. Academia, 2002.

[23] W. van der Hoek and M. Wooldridge. Cooperation, knowledge, and time: Alternating-time temporal epistemic logic and its applications. *Studia Logica*, 75(1):125–157, 2003.

[24] H. Van Ditmarsch, W. Van der Hoek, and B. Kooi. Concurrent dynamic epistemic logic for MAS. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, page 208. ACM, 2003.

[25] H. van Ditmarsch, W. van der Hoek, and B. Koot. Playing Cards with Hintikka-An Introduction to Dynamic Epistemic Logic. 2005.

[26] M. Vardi. Sometimes and not never re-revisited: on branching versus linear time. *CONCUR'98 Concurrency Theory*, pages 1–17, 1998.

[27] M. Vardi. Branching vs. linear time: Final showdown. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 1–22, 2001.

[28] M. Vardi. Linear vs. branching time: A complexity-theoretic perspective. In *Logic in Computer Science, 1998. Proceedings. Thirteenth Annual IEEE Symposium on*, pages 394–405. IEEE, 2002.

[29] I. Wegener. *Branching programs and binary decision diagrams: theory and applications*. Society for Industrial Mathematics, 2000.

[30] M. Wooldridge. *Reasoning about rational agents*. The MIT Press, 2000.

[31] K. Wu. *Discovering Formal Logic*. McGraw-Hill/Dushkin, 1994.

Selected Papers by the Author

M. Kolesár, D. Kotmanová. Applications of Model Checking to the Verification of Digital System Designs *Acta Electrotechnica and Informatica*, TU Košice,March 2009.

M. Kolesár, D. Kotmanová. Temporal Logic and Its Application in Practice. In: C.I.T. *Journal of Computing and Information Technology*, University of Zagreb, Croatia, to appear.

M. Kolesár, D. Kotmanová. Temporal Operators Until and Unless in Digital System Verification. In: *Proceedings of 10^{th} International Conference Informatics 2009*, Herl'any, November 2009.

D. Kotmanová. A Survey of Verification Methods for Program Systems. In: *Proceedings of 6^{th} Internat. Conference Informatics 2001*, pp. 283–290, Bratislava, November 2001.

D. Kotmanová. Temporal Logic of Programs. In: *Proceedings of the International Conference Cybernetics and Informatics 2002*, pp. 283-290, Trebišov, September 2002.

D. Kotmanová. Verification of a Hardware Component. In: *Proceedings of the International Conference Cybernetics and Informatics (Kybernetika a informatika)*, Michalovce, Juny 2006.

D. Kotmanová. Digital Design Verification Using Model-Checking. In: *Proceedings of 9^{th} International Conference Informatics 2007*, Bratislava, November 2007.

D. Kotmanová. Temporal Logic in Verification of Digital Circuits. *Journal of Electrical Engineering*, FEI STU, Bratislava, January 2008.

D. Kotmanová. Simulation and Verification of Digital Systems with NuSMV. In: *Acta Electrotechnica and Informatica*, TU Košice, May 2009.

D. Kotmanová. Temporal Specifications in Digital System Verification. In: *Journal of Electrical Engineering*, FEI STU, Bratislava, 2010, to appear.

D. Kotmanová. Temporal Specifications of the Digicode Model with ERROR state. : In: *Journal of Electrical Engineering*, FEI STU, Bratislava, 2010, to appear.

D. Kotmanová. Temporal Specifications of Ready-Busy Circuits. : In: *Acta Electrotechnica and Informatica*, TU Košice, 2011, to appear.

D. Kotmanová. Semantics of Temporal Operators in Digital System Verification. : In: *Journal of Applied Non-Classical Logics*, Institut de Recherche en Informatique de Toulouse (IRIT), Université Paul Sabatier, France, 2011 - to appear.