数理解析研究所講究録
第1799巻 2012年 21-28

Hybrid Automata Theoretic Specification and Verification of CPU-DRP Reconfigurable Systems

Ryo Yanase
Kanazawa University *

Gao Ying
Kanazawa University

Minami Shota
Kanazawa University

Satoshi Yamane
Kanazawa University

Abstract

In this paper, we propose formal modeling, specification and verification for CPU-DRP reconfigurable systems based on hybrid automata. First, we specify CPU and environment as real-time systems, and specify DRP as hybrid systems by using hybrid automata. Next, we verify various properties by model checking using HyTech. We have realized verification of parallel composition of CPU, DRP and environment.

1 Introduction

Embedded systems recently begin to have various functions, but increasing the number of processors causes troubles for miniaturization and saving energy. Therefore, recently, a Dynamically Reconfigurable Processor(DRP) is paid to attention[1]. In DRP, a plural number of exclusive processings is executed in the same board by dynamically changing the circuit configuration[1, 2]. DRP is used as an accelerator of CPU, and some DRP is loosely connected with the CPU[1]. In this paper, DRP is loosely connected with the CPU. We model the embedded system which integrates CPU and DRP cooperatively. Also, we specify the embedded system using hybrid automata, and verify properties by using model checker HyTech[3].

1.1 Related Works

1.1.1 Specification language

A specification language of dynamic reconfigurable system is either reactive model, real-time model or hybrid model. Also, the style is either process algebra, automaton or Petri Net. A. Deshpande has developed SHIFT[4], and F. Kratz has developed R-Charon[5] based on hybrid automaton. SHIFT[4] and R-Charon[5] have specification power for dynamically changing the structure of the network. However, as they cannot describe event trigger behaviors, then they cannot describe a dynamically reconfigurable processor. Also, the Φ-calculus is a process algebra based on hybrid reconfigurable modeling language[6]. But the Φ-calculus considers continuous behavior to be a property of an explicit environment instead of being part of other embedded systems as we do. On the other hand, thought J. Teich[7] and K.Onogi[8] have studied modeling method of DRP related to this paper based on discrete event system, their method cannot specify DRP, in which the operating frequency of DRP changes dynamically.

*ryanase@csl.ec.t.kanazawa-u.ac.jp

1.1.2 Verification

Wang Yi and co-workers have proposed the general schedulability checking problem for real-time tasks is a reachability problem for a decidable class of timed automata extended with subtraction[15]. Also, Cimatti and Palopoli have modeled real-time tasks by parametric timed automata[16]. Wang Yi's and Cimatti's work mean real-time properties such as schedulability can be verified by timed automata. In this paper, as we verify both real-time and hybrid properties, we specify DRP using hybrid automata.

1.1.3 Architecture

Pellizzoni and Caccamo have developed reconfigurable architecture composed of CPU and reconfigurable area (FPGA) with periodic tasks[17]. Also, H. Nakano and T. Shindo have developed dynamically reconfigurable processor LSI[2]. In this paper, our model is reconfigurable architecture composed of CPU and dynamically reconfigurable processor LSI(DRP). We have already specified reconfigurable system composed of CPU and DRP using hybrid automata. Moreover we have verified schedulability of specification using HyTech[11, 18].

In this paper, we improve specification and verify safety and liveness with hybrid, real-time and reactive features.

2 Model of CPU, DRP and Environment

We model the embedded system that combines CPU, DRP and environment as shown in Figure 1. The embedded system advances processing by cooperated operations of CPU and DRP. Tasks on CPU are dynamically created by an external environment. Tasks are executed under the management of CPU-Dispatcher. The task that can be executed at the same time on CPU is one. When it is necessary to process two or more tasks, the allocation of CPU is changed according to priority (preemption). When there is a description to use DRP in a task, CPU-Dispatcher outputs the creation demand of co-task of DRP to DRP-Dispatcher. At this time, initial values such as a necessary substrate area and operating frequency in co-task information are inputted into co-task. Two or more co-tasks are executable at the same time on DRP. It is arranged on the substrate as long as there is becoming empty in the tile in order of arrival. However, when you execute co-task a and b at the same time, it operates by the slowest value f_b in the operating frequency of co-tasks under execution as shown lower in Figure 2. We specify dynamic creation and destruction of tasks and co-tasks by a static system.

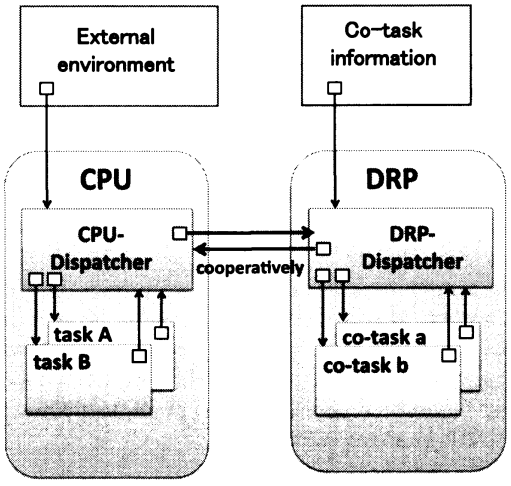

Figure 1: Overview of dynamic reconfigurable system

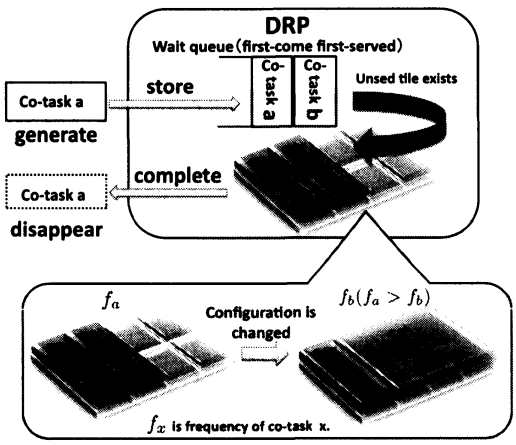

Figure 2: Behavior of dynamic reconfigurable processor

3 Specification language of DRP, CPU and Environment

We define syntax and semantics of a linear hybrid automaton [3] of specification language of DRP, CPU and environment as follows. We extend a linear hybrid automaton [3] with discrete variables.

3.1 Syntax of a linear hybrid automaton

First, the syntax of a linear hybrid automaton is formally defined.

Syntax of a linear hybrid automaton An invariable condition and a guard condition are defined as follows:

$$\phi ::= true \mid asap \mid \gamma_1 \sim \gamma_2 \mid \phi_1 \wedge \phi_2$$

, where

$$\gamma ::= x \mid d \mid c \mid \gamma_1 + \gamma_2 \mid \gamma_1 - \gamma_2$$

$\sim \in \{<, >, ==, \leq, \geq\}$, $x \in X$ is a real-valued variable, $d \in D$ is a discrete variable, c is real number. *asap* is included only in the guard condition. The transition relation to which *asap* attaches gives priority more than a timed transition in HyTech[3]. Let $B(X)$ be the set of invariable conditions and guard conditions.

A *flow*, which assigns a flow condition to each location, is the following predicate:

$$\alpha ::= \dot{x} = c$$

The dotted variable $\dot{x} \in \dot{X}$ refers to the first derivative of x with respect to time, i.e., dx/dt. Let $F(X)$ be the set of flow conditions. Also, arithmetic expression over a finite set $V(= X \cup D)$ is defined as follows:

$$upd ::= v := const \mid v := v + const$$

, where *const* is real number, integer,character string. Let $UPD(V)$ be the set of arithmetic expressions.

A linear hybrid automaton LHA is a tuple $(X, D, L, inv, init, flow, E, Act)$ that consists of the following components:

- X is a finite set of real-valued variables.
- D is a finite set of discrete variables.
- L is a finite set of locations.
- *inv* is a function that assigns an invariant condition $\phi \in B(X)$ to each location $l \in L$.
- *init* is an initial condition consists of the set of initial locations and arithmetic expressions.
- *flow* is a function that assigns a flow condition $\alpha \in F(X)$ to each location $l \in L$.
- *Act* is a finite set of actions, where $Act = Act_{in} \cup Act_{out} \cup \{\tau\}$. Here Act_{in} is a finite set of input actions, Act_{out} is a finite set of output actions, τ is an internal action.
- $E \subseteq L \times Act \times B(X) \times 2^{UPD(V)} \times L$ is a finite set called the transition relation. An element of E is a tuple of the form $\langle l, action, \phi, UPD(V), l' \rangle$, where action is either $a!$, $a?$, τ, and $UPD(X)$ is a finite set of arithmetic expressions, and ϕ is a guard condition.

Here a linear hybrid automaton LHA is a stopwatch automaton if a flow condition α is defined as follows:

$$\alpha ::= \dot{x} = 0 \mid \dot{x} = 1$$

■

3.2 Semantics of a linear hybrid automaton

First, we define a state of a linear hybrid automaton.

State of a linear hybrid automaton A state of a linear hybrid automaton is a pair (l, μ, ν) consisting of a location $l \in L$, $\nu : X \to \mathbf{R}$, $\mu : D \to \mathbf{Z} \cup \mathbf{STRING}$, where $\mathbf{Z}$ is integer, **STRING** is a set of character strings.

■

Transitions of a linear hybrid automaton consist of a timed transition and two discrete transitions. Next, we define a timed transition of a linear hybrid automaton.

Timed transition

$$(l, \mu, \nu) \xrightarrow{\delta} (l, \mu, \nu')$$

Here a curve of *flow* is a differentiable function $f : [0, \delta] \to \mathbf{R}^n$, where $|X| = n$, $f(0) = \nu, f(\delta) = \nu'$.

■

Next, discrete transitions consist of an internal transition and a synchronization transition.

An internal transition

$$(l, \mu, \nu) \xrightarrow{\tau, guard, UPD(V)} (l', \mu', \nu')$$

ϕ is assigned to *guard*, variables are updated by $UPD(V)$. Also, if $\phi = asap$, then the discrete transition is immediately done.

■

Synchronization transition When automaton 1, automaton 2, and automaton 3 change synchronously by action $a?$ and $a!$, the behaviors are formally defined as follows:

$$(l_1, \mu_1, \nu_1) \xrightarrow{a!, guard_1, UPD(V_1)} (l_1', \mu_1', \nu_1'),\ (l_2, \mu_2, \nu_2) \xrightarrow{a?, guard_2, UPD(V_2)} (l_2', \mu_2', \nu_2'),$$
$$(l_3, \mu_3, \nu_3) \xrightarrow{a?, guard_3, UPD(V_3)} (l_3', \mu_3', \nu_3')$$

Automaton 1 outputs $a!$, then both automaton 2 and automaton 3 input $a?$.

■

Finally, we define a run of a linear hybrid automaton.

A run of a linear hybrid automaton

$$(l_0, \mu_0, \nu_0) \xrightarrow{\delta_1} (l_0, \mu_0, \nu_1) \xrightarrow{e_1} (l_1, \mu_1, \nu_2) \cdots$$

, where l_0 is an initial location, both ν_0 and μ_0 are valuations given by an initial condition, $e_1 \in E$ is a transition relation.

■

3.3 Parallel composition

The communications between external environment, CPU-Dispatcher, task, DRP-Dispatcher and co-task in Figure 1 are expressed by parallel compositions of hybrid automata. For given

$$LHA_i = (X_i, D_i, L_i, inv_i, init_i, flow_i, E_i, Act_i)\ \ (i = 1, \ldots, n),$$

the parallel composition $LHA_1 \times \cdots \times LHA_n$ is $LHA = (X, D, L, inv, init, flow, E, Act)$ consisting of the following components:

- $X = X_1 \cup \cdots \cup X_n$ is a finite set of variables.

- $D = D_1 \cup \cdots \cup D_n$ is a finite set of discrete variables.
- $L = L_1 \times \cdots \times L_n$ is a finite set of locations.
- inv is a function that assigns an invariant condition $\phi \in B(X)$ to each location $l \in L$, where $\phi = \phi_1 \wedge \cdots \wedge \phi_n$, $\phi_1 \in B(X_1), \ldots, \phi_n \in B(X_n)$.
- $init = (init_1, \ldots, init_n)$ is an initial condition.
- $flow$ is a function that assigns a flow condition $\alpha \in F(X)$ to each location $l \in L$, where $\alpha = \alpha_1 \wedge \cdots \wedge \alpha_n$.
- $Act = \{\tau\}$ is a finite set of actions. The input action synchronizes with the output action and it becomes an internal action τ.
- $E \subseteq L \times Act \times B(X) \times 2^{UPD(V)} \times L$ is a finite set called the transition relation. An element of E is a tuple of the form $\langle l, \tau, \phi, UPD(V), l' \rangle$, where an element of E is a tuple of the form $\langle l, \tau, \phi, UPD(V), l' \rangle$, where $l, l' \in L, \tau \in Act, \phi = \phi_1 \wedge \cdots \wedge \phi_n$, $UPD(V) = UPD(V_1) \cup \cdots \cup UPD(V_n)$.

4 Configuration of CPU, DRP and Environment

We show configuration of CPU, DRP and environment based on hybrid automata in Figure 3.

1. *Ext* is an automaton which expresses the environment. *Ext* sends a create demand on *Task*.
2. *Task* is an automaton that changes to a ready state from a NONE state when an activation demand is received from *Ext*, and a dispatch demand is sent to *Task Dispatcher*. It changes to an executing state when selected by *Task Dispatcher* as an execution task. An executing task might send processing activities to a *Co-task*. In this case, the task changes to a waiting state, the dispatch demand is sent to the *Task Dispatcher* and the processing demand of the *Co-task* is sent to the *Co-task*. It changes to a waiting state when the end response of the processing *Co-task* is returned. When the executing task ends processing, the dispatch demand is sent to *Task Dispatcher*.
3. *Task Dispatcher* is an automaton for dispatching tasks. When a dispatch demand is received from a task, the task with the highest priority changes to the executing state, and other tasks change to waiting states. There is also a dispatch demand from *Task*.
4. *DRP Dispatcher* is an automaton for dispatching *Co-task* in DRP. When *DRP Dispatcher* receives dispatch demand from *Co-Task*, it sends an execution demand to *Co-task* with the highest priority. if there are tiles for executing a *Co-task* of a head of waiting queue.
5. *Co-Task* is an automaton of a co-task executing on DRP. It changes to a ready state when a create demand of *Co-task* is received from *Task*. It changes to the executing state when an execution demand is received from *DRP Dispatcher*, and the processing of *Co-task* begins. The processing end response is returned to *task* when processing ends.
6. *DRP Frequency* is an automaton that manages the frequency of DRP. When a *Co-task* is executed with slow operating frequency, the inclination of the execution time of *Co-task* under execution is changed.

5 Verification for CPU-DRP reconfigurable systems

5.1 Verification Properties

5.1.1 Overview

Dynamically reconfigurable systems have the following three significant features that are called hybrid, real-time and reactive features[10, 19, 20].

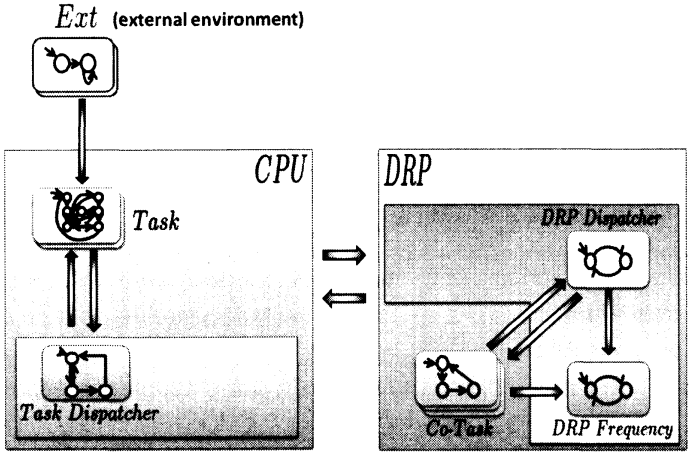

Figure 3: Configuration of CPU, DRP and environment

- Hybrid feature——Systems behave as a hybrid system with dynamically changing the operating frequency of DRP
- Real-time feature——Systems behave as a real-time system with the deadline in the processing task
- Reactive feature——Systems behave as a reactive system with responding to the input from the environment.

We must verify whether our specification satisfies safety and liveness requirements[21] with the above features or not. Therefore, we classify the verification properties into six categories shown in Table 1.

Properties with hybrid feature In this paragraph, we explain liveness and safety requirements with hybrid feature.

Liveness requirement with hybrid feature is a requirement of "idle frequency." All co-tasks are destroyed and the operating frequency of DRP becomes the idle frequency in the future. Safety requirement with hybrid feature is a requirement of "minimum frequency." The operating frequency of DRP is always fixed to a minimum frequency of running co-tasks.

Properties with real-time feature Next, we explain liveness and safety requirements with real-time feature.

Liveness requirement with real-time feature is a requirement of "dispatch of co-tasks." If a co-task is called by a task, it is executed within the maximum waiting time, where the maximum waiting time is a difference between the deadline and CPU computation time.

Safety requirement with real-time feature is a requirement of "CPU schedulability" for tasks[9]. The remaining time needed to finish executing a task is always less than the remaining time until the deadline.

Properties with reactive feature Finally, we explain liveness and safety requirements with reactive feature.

Liveness requirement with reactive feature is a requirement of "destruction of co-tasks". If the co-task is dispatched by the DRP dispatcher, it is destroyed in the future.

Safety with reactive feature is a requirement of "control of tiles" for management of resource. In this paper, we assume that the total number of tiles is 8. Therefore, the number of unused tiles always ranges from 0 to 8.

5.2 Practical verification experiment

Using monitor automata, we have verified six properties for the dynamically reconfigurable system by using HyTech.

Now, we show the case of CPU-DRP reconfigurable system that has two tasks and two co-tasks. Table 2 shows parameters of tasks. A period T_A of task A is 70 milliseconds and a period T_B of task B is 200 milliseconds. Processing procedure "20 ms, Co_Task_a, 10 ms, Co_Task_b" means that "co-task a is called after CPU advances processing for 20 milliseconds, and co-task b is called after CPU advances processing for 10 milliseconds afterwards". Also, parameters of co-tasks are shown in Table 3.

For our experimental environment, we have used the machine which runs CentOS version 6.0 with Intel Core 2 Quad 2.50GHz processor and 8GB RAM. Table 4 shows the results of the verification experiment. In this case, all requirements are satisfied and can be computed with less than 600 MB RAM in less than 15 seconds of CPU time. For requirements with hybrid feature, the requirement of idle frequency can be verified with 248 MB RAM in 10.23 seconds, and the requirement of minimum frequency can be verified with 237 MB RAM in 9.67 seconds. For requirements with real-time feature, the requirement of dispatch of co-tasks can be verified with 105 MB RAM in 4.16 seconds, and the requirement of CPU schedulability can be verified with 545 MB RAM in 14.99 seconds. For requirements with reactive feature, the requirement of destruction of co-tasks can be verified with 24 MB RAM in 1.82 seconds, and the requirement of control of tiles can be verified with 134 MB RAM in 5.32 seconds. By verification, it is not possible to schedule the system because the remainder time until the deadline became 29 at time 171 though the time of 30 is needed for CPU processing of task B. In this case, required memory and computation time are 211MB and 7.1 seconds.

Table 1: Classification of properties

	Hybrid feature	Real-time feature	Reactive feature
Liveness requirement	Idlie frequency	Dispatch of co-tasks	Destruction of co-tasks
Safety requirent	Minimum frequency	CPU schedulability	Control of tiles

Table 2: Task parameter

Task	Period	Deadline	Priority	Processing procedure
A	70 ms	70 ms	high	20 ms, Co_Task_a, 10 ms, Co_Task_b
B	200 ms	200 ms	low	Co_Task_a, 97 ms

Table 3: Co-task information

Co-Task	Computation time	Deadline	Tiles	RF†
a	10 ms	15 ms	2	1
b	5 ms	10 ms	6	1/2

†RF – Ratio of Frequency

Table 4: Experimental results

Requirement	Satisfied or NOT	Memory	Time
Idle frequency	Satisfied	248 MB	10.23 s
Minimum frequency	Satisfied	237 MB	9.67 s
Dispatch of co-tasks	Satisfied	105 MB	4.16 s
CPU schedulability	Satisfied	545 MB	14.99 s
Destruction of co-tasks	Satisfied	24 MB	1.82 s
Control of tiles	Satisfied	134 MB	5.32 s

6 Conclusions

In this article, we have proposed formal modeling, specification of CPU-DRP reconfigurable system based on hybrid autotama, and have verified six properties such as safety and liveness requirements with hybrid, real-time and reactive features using an existing model checker HyTech[3]. Especially we specify creation and destruction of tasks and co-tasks by a static model. But in fact, co-tasks on DRP are created and destroyed. And thus the state-space explosion problems may occur in verification stages. In order to avoid this problem, we have already developed dynamic hybrid automaton and its dynamic hybrid

CEGAR(CounterExample-Guided Abstraction Refinement)[14], and we are now implementing dynamic hybrid CEGAR verifier.

References

[1] H. Amano, A Survey on Dynamically Reconfigurable Processors, *IEICE Transactions*, 89-B(12), pp.3179-3187, 2006.

[2] H. Nakano, T. Shindo, T. Kazami and M. Motomura, Development of Dynamically Reconfigurable Processor LSI, *NEC Technical Journal*, Vol.56(4), pp.99-102, 2003.

[3] T.A.Henzinger and P.Ho, H.Wong-Toi, HyTech: A Model Checker for Hybrid Systems, *STTT*, Vol.1(1-2), pp.110-122, 1997.

[4] A.Deshp, A.Gollu and L.Semenzato, The SHIFT programming language and run-time system for dynamic networks of hybrid systems, *IEEE Transactions on Automatic Control*, Vol.43(4), pp.584-587, 1998.

[5] F.Kratz, O.Sokolsky, G.J.Papspas and I.Lee, R-Charon, a Modeling Language for Reconfigurable Hybrid Systems, *LNCS 3927*, pp.392-406, 2006.

[6] W.C. Rounds and H. Song, The Phi-Calculus: A Language for Distributed Control of Reconfigurable Embedded Systems, *LNCS 2623*, pp.435-449, 2003.

[7] J.Teich and M.Koster, (Self-)reconfigurable finite state machines: Theory and implementation, *Proc. of Design, automation and test in Europe*, pp.559–568, 2002.

[8] K.Onogi and T.Ushio, Scheduling of Periodic Tasks on a Dynamically Reconfigurable Device Using Timed Discrete Event Systems, *IEICE Transactions*, E89-A(11), pp.3227–3234, 2006.

[9] J. Goossens and P. Richard, Overview of real-time scheduling problems, *Proceedings of ninth international workshop on project management and Scheduling*, pp.13-22, 2004.

[10] P.Marwedel, Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems,2nd edition, Springer, 2010.

[11] S.Takinai and S.Yamane, Case study of Modeling, Specification and Finite Model Checking for Preemptive Embedded Software, *IEICE Transactions*, E93-D(11), pp.2403–2415, 2010. (in Japanese)

[12] M.Y. Vardi and P. Wolper, An automata-theoretic approach to automatic program verification, *Proceedings of Logic in Computer Science*, pp.322-331. IEEE Computer Society Press, 1986.

[13] T.A. Henzinger, Pei-Hsin Ho and Howard Wong-Toi, A user guide to HyTech, *LNCS 1019*, Springer, pp. 41-71, 1995.

[14] M.Sakai and S.Yamane, Dynamic hybrid hybrid automaton and Dynamic hybrid CEGAR, *RIMS Kokyuroku*, RIMS(Kyoto University) 1744, pp.15-24, 2011. (in Japanese)

[15] E. Fersman, P. Pettersson, and W. Yi, Timed automata with asynchronous processes: Schedulability and decidability, *LNCS 2280*, pp.67-82, 2002.

[16] A. Cimatti, L. Palopoli and Y. Ramadian, Symbolic Computation of Schedulability Regions Using Parametric Timed Automata, *IEEE Real-Time Systems Symposium*, pp.80-89, 2008.

[17] R. Pellizzoni and M. Caccamo, Hybrid Hardware-Software Architecture for Reconfigurable Real-Time Systems, *Proceedings of the 14th IEEE RTAS*, pp.273-284, 2008

[18] S. Minami, S. Takinai, S. Sekoguchi, Y. Nakai and S. Yamane, Modeling, Specification and Model checking of dynamically reconfigurable processors, *Computer Software 28(1)*, pp.190-216, 2011.

[19] Z. Manna and A. Pnueli, Models for Reactivity, *Acta Inf. 30(7)*, pp.609-678, 1993.

[20] Hideharu Amano and Yoshinori Adachi and Satoshi Tsutsumi and Kenichiro Ishikawa, A context dependent clock control mechanism for dynamically reconfigurable processors, *Technical Report of IEICE*, 104(589), pp.13-16, 2005.

[21] E. M. Clarke, Jr., O. Grumberg and D. A. Peled, Model Checking, The MIT Press, 1999.