Automatic Control and Computer Sciences
Vol.36, No.6, pp.40-47, 2002

Avtomatika i Vychislitel'naya Tekhnika
Vol.36, No.6, pp.46-55, 2002

RACES OF ACTIONS IN PARALLEL SYSTEMS

A. PETRENKO, *Professor, Sr.Dr. (Computer Sciences)*
Centre de Recherche Informatique de Montreal
550 Sherbrooke West, Suite 100
Montreal H3A 1B9 (Canada)

A.ULRICH, *Dr.Ing.*
Siemens AG, Corporate Technology
ZT SE 1, 81730 Munich (Germany)

V.P. CHAPENKO, *Senior Researcher, Dr.Sc. (Computer Sciences)*
Institute of Electronics and Computer Science
Latvian University
Dzerbenes 14, LV-1006, Riga (Latvia)

A method of detecting races of actions in a parallel system based on analysis of its specification given in the form of systems of labeled transitions is considered. A classification of the race conditions of actions is proposed. Three types of environments in which a system may function are identified. A method of detecting races in a system functioning in a slow-sequential environment that permits execution of an action by the system only in its stable states is investigated.

1. INTRODUCTION

Races may be one of the causes for the nondeterministic behavior of concurrent systems. Situations that are not enabled in the functioning of a concurrent system which is at the design stage or in a system that already exists may arise as a result of races. Races may lead to disturbances in the functioning algorithm of a concurrent system. Problems of testing concurrent systems in which races are possible are specific to each system. It is necessary to consider faults that are due to different types of races, bearing in mind the features of the internal interaction between the components of the concurrent system as well as the interaction of the system with its environment. A test architecture and a tester that interacts with the system must not introduce additional races into a system that is being tested.

The concept of races has been studied in the theory of sequential circuits and finite-state machines [1–4]. Depending on the causes for their appearance, races are usually divided into logic hazards and functional races. Logic hazards in a combinatorial circuit are caused by the scatter of the natural delays of the components (logical elements and junction routes) in the circuit. During a transition process in the circuit these races manifest themselves at the outputs of the circuit in the form of short-term false pulse signals. In a sequential circuit, however, these false signals may propagate through the feedback circuits and may be stored, as a result of which internal states may be replaced in unforeseen ways, and the states of the output may be false. Functional races in the sequential circuit of a finite-state automaton are due to the fact that variables which, according to the performance conditions of the automaton, must vary strictly simultaneously, in fact do not vary simultaneously in a realization of the automaton. The name given to these races underscores their dependence on the properties of the Boolean functions which are implemented, but not on the properties of the realizing circuit and the characteristics of their elements. The combinatorial part of a circuit that realizes the automaton is investigated in analyzing these races. The input variables of the automaton and the internal variables (i.e., the variables of the outputs of the memory elements) are correlated with the inputs of the circuit while the output variables of the automaton and the internal variables that control the memory elements, with the outputs of the circuit. Functional races in sequential circuits may arise only in

those situations in which two or more variables belonging to the set of input and internal variables vary simultaneously. This condition is called the race condition [2, 3, 4]. In terms of the types of variables that vary simultaneously, functional races are usually divided into three types [1, 2]: races of input variables, races of internal variables, and races of input and internal variables. Identification of these races is realized through the analysis of the race conditions from the transition table of the finite-state automaton.

In general, races are the source of faults of concurrent software systems that are difficult to detect in the course of execution of a test. It is usually necessary to repeatedly execute test runs of one and the same test with variable rate of execution in order to detect such faults. One alternative to such testing is to test potential races on the basis of a formal specification of the concurrent system that is created in the course of its design. No attention is paid to race conditions in general methods of verification, however, since in these methods all possible executable sequences of the system are evaluated in systematic fashion [5]. Consequently, a technique for the detection of race conditions must be developed.

Studies on the analysis of races were first undertaken in the context of sequential circuits [6]. In [6] the concept of the difference between a race and a race condition was introduced, understanding a race condition as leading to a race if a circuit attains different final states in the course of its functional activity. The appearance of races is explained by the specification of the circuit.

Analysis of races in concurrent software is focused principally on the analysis of routes for implementation of a concurrent system. Thus, the method described in [7] does not suggest analysis of the specification of a concurrent system, instead it is suggested that whether a race condition exists should be concluded on the basis of the realized routes. The test designer decides whether some set of message exchanges will, in fact, produce races. Analysis of a route is based on identifying the relations between messages that have been encountered previously [8]. Such an approach is also presented in [9]. Another method proposes that races be detected in the course of the execution of a concurrent program, using for this purpose a probe program tool [10]. One drawback of these methods is that the races remain undetected if the required route is not executable. A more systematic method was presented in [11]. The latter study discussed the detection of potential races on the basis of an analysis of the behavior of the system. In this method, different lines of communication are used, similar to asynchronous transfer with or without queues.

A collection of interacting labeled transition systems (LTS) [12] is a commonly employed model of a concurrent system. In such a model, the components of the concurrent system are described in the form of corresponding systems of labeled transitions that interact in synchrony with each other. A labeled transition system is characterized by a set of states, including the initial state, a set of actions, and a set of transitions. The concept of an action does not distinguish between the event at the input of a component and the activity at its output, in fact, an action may denote the production or transmission of a message as well as, for example, execution of a local operation. The term, transition, denote a change in the states of the labeled transition system induced by an action. An action a is resolvable in state s if a transition from state s into state s' induced by a is stated in the specification of the system. The operation of the components of a concurrent system is modeled by sequences of states and corresponding actions. A labeled transition system may be represented in the form of an oriented graph whose vertices are states and whose arcs are labeled transitions. The label on an arc is the name of the action that induces the transition. A labeled transition system serves as a semantic model of a number of languages used in the specification of concurrent systems (LOTOS, Language of Temporal Ordering Specification; CCS, Calculation for Communicating Systems; CSP, Communicating Sequential Processes). In the present article, an approach to the detection of races in a concurrent system that is based on analysis of the specification, represented in the form of a collection of labeled transition systems, is considered. To distinguish races in a concurrent system specified in the form of interacting labeled transition systems from functional races of variables in sequential circuits and in finite-state automata, races in concurrent systems will be referred to, as in [13], as action races. By an action race condition we will understand in the present article the situation in which more than one action is enabled in some global state of the concurrent system. If the order of execution of enabled actions is arbitrary, it may turn out that a global state of a concurrent system attained as a result of a transition may not be unique. The present article is based in part on material set forth in summary in [13]. A classification of race conditions by type of system action is proposed. Three different types of environments in which the system functions are distinguished. A method of detecting races in a system that functions in a sequential-slow environment and which processes a single action assigned to the system only when it is found in a stable state is presented.

2. ASSUMPTIONS AND DEFINITIONS

A concurrent system includes a set of independent components that interact with each other by means of message transfers. The inputs and outputs of a component are not distinguished. Each component has ports for synchronous interaction with the environment or with other components. A concurrent system is an open system, that is, it maintains interaction with the environment.

Formally, a concurrent system $\Im$ is modeled by a set of n finite labeled-transition systems that interact synchronously according to a rendezvous method. A rendezvous is an indivisible operation of interaction between two components that is executed only in the case in which both components are ready for this interaction. Both two-sided rendezvous as well as multi-sided rendezvous will be considered subsequently. Each component of a concurrent system is represented by a corresponding labeled-transition system. Transfer of a message and its reception through the ports are referred to as actions in a labeled-transition system.

Definition 1. A labeled-transition system (LTS) M is defined by the quadruple

$$(S, A \cup \{\tau\}, \rightarrow, s_0), \tag{1}$$

where S is a finite set of states; A, a finite set of actions; $s_0 \in S$, initial state; τ, unobservable action of the system M; and $\rightarrow \subseteq S \times (A \cup \{\tau\}) \times S$, transition relation.

A transition $(s_1, a, s_2) \in \rightarrow$ is also written $s_1 - a \rightarrow s_2$. Suppose that the notation $s = a \Rightarrow s'$ denotes the sequence

$$s - \tau \rightarrow \ldots - a \rightarrow \ldots - \tau \rightarrow s'. \tag{2}$$

Moreover, let $s = \sigma \rightarrow s'$ express the sequence

$$s = a_1 \Rightarrow s_1 = a_2 \Rightarrow s_2 = \ldots \Rightarrow s', \tag{3}$$

where $\sigma = a_1 a_2 \ldots$. We will suppose that every labeled-transition system is initially connected, that is, that for each state s' there exists a route σ such that $s_0 = \sigma \Rightarrow s'$. Since in the general case, a labeled-transition system is nondeterministic, state s', which is attained after route σ is executed, may not be unique. Therefore, the expression (s **after** σ) denotes the set

$$\{s' \mid s = \sigma \Rightarrow s'\}. \tag{4}$$

Definition 2. A labeled-transition system M is deterministic if

$$|(s \textbf{ after } \sigma)| \leq 1 \quad \text{for all} \quad \sigma \in A^* \quad \text{and} \quad s \in S. \tag{5}$$

Below, concurrent systems consisting of deterministic systems M with action τ will be considered. Their behavior is characterized in terms of routes. A set of routes of state s may be represented in the form

$$\{\sigma \in A^* \mid \exists\, s' \in S:\ s = \sigma \Rightarrow s'\} = \mathbf{Tr}(s), \tag{6}$$

and a set of routes of M as $\mathbf{Tr}(s_0)$.

The composition of two deterministic systems P and Q is expressed in a composite machine by application of the composition operator:

$$\|: \mathbf{LTS}(A_P) \times \mathbf{LTS}(A_Q) \rightarrow \mathbf{LTS}(A_P \cup A_Q), \tag{7}$$

where A_P, respectively, A_Q is a set of actions in system P and Q, respectively; and $\mathbf{LTS}(A_P)$, respectively, $\mathbf{LTS}(A_Q)$, a set of all possible labeled-transition systems on the set of actions A_P, respectively, A_Q.

Definition 3. Suppose we are given two deterministic labeled-transition systems

$$P = (S_P, A_P, \to_P, s_P), \quad Q = (S_Q, A_Q, \to_Q, s_Q). \tag{8}$$

The concurrent composition operator $\|$ is defined by the following inference rules:

- **If** $P - a \to P'$, $a \notin A_Q$ **then** $(P \| Q) - a \to (P' \| Q)$.
- **If** $Q - a \to Q'$, $a \notin A_P$ **then** $(P \| Q) - a \to (P \| Q')$. (9)
- **If** $P - a \to P'$, $Q - a \to Q'$, **then** $(P \| Q) - a \to (P' \| Q')$.

The operator correlates with a given concurrent system a composite machine the states of which are referred to as global states, in contrast to the states of a labeled-transition system, which are referred to as local states.

Definition 4. By the composite machine

$$C_\Im = M_1 \| \ldots \| M_n \tag{10}$$

of a concurrent system $\Im$ represented by n labeled-transition systems of the form

$$M_i = (S_i, A_i, \to_i, s_{0i}) \tag{11}$$

is understood the labeled-transition system

$$(S_\Im, A_\Im, \to_\Im, s_{0\Im}), \quad s_{0\Im} = (s_{01}, \ldots, s_{0n}), \tag{12}$$

where $s_{0\Im}$ is the initial global state and

$$S_\Im \subseteq S_1 \times \ldots \times S_n; \quad A_\Im \subseteq A_1 \cup \ldots \cup A_n; \quad \to_\Im \subseteq S_\Im \times A_\Im \times S_\Im \tag{13}$$

are the smallest sets that may be obtained as a result of application of the operator $\|$.

For the sake of simplicity, we will assume that

$$A_\Im = A_1 \cup \ldots \cup A_n. \tag{14}$$

Let n be a set of indices $\{1, \ldots, n\}$ assigned to each labeled-transition system in the concurrent system $\Im$. For the action $a \in A_\Im$, the notation $i(a)$ denotes the set of occurrences $\{i \in N \mid a \in A_i\}$ of the action a, that is, the set of indices of labeled-transition systems that are implicated in a rendezvous relative to action a.

Let us assume that there are no isolated components in the concurrent system $\Im$, i.e., that at least one action a such that $|i(a)| > 1$ is assigned to each component. Suppose a subset of actions $A_e \subseteq A_\Im$ is given. Actions that belong to A_e are said to be external if

$$A_e \supseteq \{a \in A_\Im \mid |i(a)| = 1\}. \tag{15}$$

Off-line systems having an empty set of external actions will not be considered in the present article. It is assumed that $A_e \neq \varnothing$. Thus, the concurrent system which is being considered here is an open system, that is, it requires an environment for the execution of external actions. The set of actions in the system $\Im$ that do not belong to A_e is the set of internal actions $A_i = A_\Im \setminus A_e$ of the system $\Im$. We also introduce the concepts of internal and external transitions, which are realized under the influence of internal, respectively, external actions. Suppose

$$\mathbf{enabled}(s) = \{a \mid \exists\, s' \in S_\Im : (s, a, s') \in \to_\Im\} \tag{16}$$

denotes the set of actions that belong to global transitions and are enabled in the global state s. A stable global state is a state s in which

$$\mathbf{enabled}(s) \subseteq A_e; \tag{17}$$

a transient global state is a state in which

$$\mathbf{enabled}(s) \neq \varnothing \quad \text{and} \quad \mathbf{enabled}(s) \not\subset A_e. \tag{18}$$

A deadlock state is a global state in which there are no enabled actions,

$$\mathbf{enabled}(s) = \varnothing. \tag{19}$$

In the present article it is assumed that a concurrent system has at least one stable global state (one initial state). Such an assumption is meant to exclude oscillatory systems.

3. CLASSICAL RACE CONDITIONS

Intuitively, in the model of synchronous interaction which we are considering an action race condition exists whenever several actions are enabled in some global state. A race condition leads to a race if in any other orders of execution of these actions, the system reaches different finite (stable) states. This phenomenon recalls the phenomenon of (critical, dangerous) races in sequential circuits [6]. Depending on the types of actions, we will distinguish three types of race conditions:

- race condition of internal actions, or *i*-race conditions (*i*-race)
- race conditions of external actions, or *e*-race conditions (*e*-race)
- conditions of mixed (internal and external) races, or *m*-race conditions (*m*-race)

Only *i*-race conditions may exist in off-line (closed) systems. The presence of an individual type of race condition in a concurrent (open) system depends on which type of environment the particular system is embedded in. We distinguish three basic types of environments depending on the ability of the environment to process a particular action or several actions during the time period in which the system is found in a stable state or in either a stable or an unstable state.

- environment that processes a single external action only when the concurrent system is in a stable state, called a sequential-slow environment
- environment which may process isolated external actions sequentially even before the concurrent system attains a succeeding stable state, called a sequential-fast environment; in addition to *i*-race conditions, *m*-race conditions occur in a system that is found within such an environment
- finally, if the environment is able to process several external actions simultaneously, starting from a point when the system is in a stable state, it is called a concurrent environment; note that since these external actions may be executed in any possible sequence, a concurrent environment is also a sequential-fast environment; *e*-race conditions may exist in such an environment

4. SEQUENTIAL-SLOW ENVIRONMENT

Before transmitting a succeeding external action to a concurrent system, a sequential-slow environment is in a state of expectation until it completes execution of internal actions. Analysis of the behavior of a concurrent system controlled by this environment may be greatly simplified if discussion is focused not only on the composite machine (Definition 4), but rather on its submachine. To construct such a submachine we introduce the operator $\rfloor\!\!\lfloor$ for modified decomposition of two labeled-transition systems P and Q with fixed set of external actions $A_e \subseteq A_P \cup A_Q$. Such a decomposition constitutes a map

$$\rfloor A_e \lfloor\!\!: \mathbf{LTS}(A_P) \times \mathbf{LTS}(A_Q) \to \mathbf{LTS}(A_P \cup A_Q). \tag{20}$$

By comparison with the ordinary concurrent composition operator $\|$, the new operator suppresses execution of external actions whenever the system is able to execute an internal action.

Definition 5. Suppose we are given two systems P and Q of the form (8) and sets A_e and A_i of external and internal actions, respectively, and let

$$A_e \cup A_i = A_P \cup A_Q. \tag{21}$$

The A_e-slow composite operator $\rfloor A_e \lfloor$ is specified by the following inference rules. If there exists an action $a \in A_i$ such that

$$P - a \to P', \quad Q - a \to Q', \text{ then} \tag{22}$$

$$(P \rfloor A_e \lfloor Q) - a \to (P' \rfloor A_e \lfloor Q'), \text{ else} \tag{23}$$

- **if** $P - a \to P'$, $a \notin A_Q$ **then** $(P \rfloor A_e \lfloor Q) - a \to (P' \rfloor A_e \lfloor Q)$, (24)
- **if** $Q - a \to Q'$, $a \notin A_P$ **then** $(P \rfloor A_e \lfloor Q) - a \to (P \rfloor A_e \lfloor Q')$, (25)
- **if** $P - a \to P'$, $Q - a \to Q'$ **then** $(P \rfloor A_e \lfloor Q) - a \to (P' \rfloor A_e \lfloor Q')$. (26)

This operator correlates with a concurrent system the succeeding composition.

Definition 6. Suppose we are given a concurrent system $\Im$ consisting of n labeled-transition systems M_i of the form (11) with set of external actions A_e. A slow-composite machine

$$Sc_\Im = M_1 \rfloor A_e \lfloor \ldots \rfloor A_e \lfloor M_n \tag{27}$$

of the concurrent system $\Im$ is a labeled-transition system

$$(S_{Sc}, A_{Sc}, \to_{Sc}, s_{0Sc}), \tag{28}$$

where $s_{0Sc} = s_{0\Im}$ is the initial stable state,

$$S_{Sc} \subseteq S_1 \times \ldots \times S_n, \quad A_{Sc} \subseteq A_1 \cup \ldots \cup A_n, \quad \text{and} \quad \to_{Sc} \subseteq S_\Im \times A_\Im \times S_\Im \tag{29}$$

is the smallest set that can be obtained by application of the operator $\rfloor A_e \lfloor$ to the concurrent system.

For the sake of simplicity, we will assume that $A_e \subseteq A_{Sc}$. The next assertion establishes a relationship between the two types of composite machines.

Proposition 1. Suppose we are given a concurrent system $\Im$ and its composite machine $C_\Im$. A slow composite machine $Sc_\Im$ is a submachine of $C_\Im$ for any set of external actions A_e. If $A_e = A_\Im$, i.e., if there are no external actions in the system, the two machines $Sc_\Im$ $C_\Im$ coincide.

Suppose (s **after-ext** a) denotes the set of all possible stable global states that may be attained by $\Im$ after external action a has been executed in the stable global state s, i.e., let

$$(s \textbf{ after-ext } a) = \{s' \mid \exists\, \sigma \in A_i^* : s' \in (s \textbf{ after } a\sigma) \wedge \textbf{enabled}(s') \subseteq A_e\}. \tag{30}$$

Definition 7. A stable global state s and external action a belonging to the set of actions (16) enabled in s are given. Then:

- system $\Im$ is divergent, i.e., it has livelock for a in s if

$$(s \textbf{ after-ext } a) = \varnothing; \tag{31}$$

- action a creates a race in s if

$$|(s \textbf{ after-ext } a)| > 1; \tag{32}$$

- action a is race-free in s if

$$|(s \textbf{ after-ext } a)| = 1. \tag{33}$$

System $\Im$ is race-free within a sequential-slow environment if in every stable state s each action $a \in$ **enabled**(s) is race-free. Henceforth, we will consider only convergent systems, i.e., systems that are trap-free. All races in such systems, found within a sequential-slow environment, may be characterized by means of a stable composite machine specified in the following way.

Definition 8. Suppose we are given a concurrent system $\Im$ with slow composite machine $Sc_{\Im}$. A stable composite machine is a labeled-transition system

$$St_{\Im} = (S_{St}, A_{St}, \rightarrow_{St}, s_{0St}), \tag{34}$$

where

$$s_{0St} = s_{0Sc}, \quad S_{St} \subseteq S_{Sc}, \quad A_{St} \subseteq A_e, \quad \text{and} \quad \rightarrow_{St} \subseteq S_{St} \times A_e \times S_{St} \tag{35}$$

are the smallest sets that may be obtained by application of the following rule:

$$(s, a, s') \in \rightarrow_{St} \quad \Leftrightarrow \quad s' \in (s \textbf{ after-ext } a). \tag{36}$$

For the sake of simplicity, we will assume that $A_{St} = A_e$. A stable composite machine may be obtained from a slow composite machine by means of the equivalence transformation described in [13]. If as a result of the transformation a nondeterministic stable composite machine is obtained, this will mean that there exist races in the concurrent system $\Im$ controlled by a sequential-slow environment.

Proposition 2. A concurrent system $\Im$ is race-free within a sequential-slow environment if and only if its stable composite machine $St_{\Im}$ is deterministic.

Any sequence of actions of a deterministic stable composite machine is race-free if its sequence is executed in a sequential-slow environment.

In the worst case, the number of states of a stable composite machine may attain the number of states of a composite machine. The question arises, is it possible to detect races without having to build a complete (stable) composite machine? To test whether there exists i-race conditions, it is not necessary to completely store the whole stable composite machine, since each combination of an external action and a stable state may be tested separately.

In certain cases the absence of i-race conditions may be established on the basis of the following observation. Internal actions that are simultaneously enabled in a global state do not lead to races if these actions relate to different local conditions, since the ultimate state is independent of the order of execution of these actions. Only the internal actions that are enabled in some local state may produce races.

Proposition 3. A concurrent system $\Im$ is race-free within a sequential-slow environment if there exists in this environment no more than one internal transition from any local state.

It follows from this proposition that this type of race is not controlled perfectly by the environment, despite its slow sequential behavior. If this specification has races within a sequential-slow environment, it may be concluded that there exists some design error (in other words, the specification possesses an unintended nondeterminism [7]) or the specification is, in fact, abstracted from the particular design and must be improved.

5. CONCLUSION

The method of analysis of the specification of a concurrent system that has been considered here makes it possible to detect races even in the course of design, verification, and testing of a system. However, it is not possible to entirely eliminate race conditions from a system design. Nevertheless, the prerequisites for their appearance must be found in order to take preventive steps in the system implementation. The article has discussed three different assumptions concerning the environment of a concurrent system. The assumption that there exist race conditions in a sequential-slow environment only within internal actions is very strict. A sequential-fast environment that may successively perform individual external actions until the system attains a succeeding stable state is more reasonable. Here mixed race conditions may arise. In a system controlled by a concurrent environment, there may exist, in addition, race conditions between external actions. Through application of the present approach to the analysis of specifications, it was also possible to investigate the behavior of a concurrent system with races in cases of sequential-fast and concurrent environments. However, these results do not fall within the scope of the present article.

REFERENCES

[1] E.A. Yakubaitis, A.Yu. Gobzemis, A.F. Petrenkc, and G.F. Fritsnovich, "On synthesis of asynchronous finite automata," Tekhnicheskaya Kibernetika, no. 6, pp. 139–148, 1972.
[2] A.L. Gurtovtsev, A.F. Petrenko, and V.P. Chapenko, Logic Design of Automatic Devices [in Russian], Zinatne, Riga, 1978.
[3] S. Unger, Asynchronous Sequential Circuits [Translated from English], Nauka, Moscow, 1977.
[4] S.H. Unger, Asynchronous Sequential Switching Circuits, Wiley-Interscience, New York, 1969.
[5] P. Godefroid, "Partial-order methods for the verification of concurrent systems," Lecture Notes in Computer Science 1032, Springer, 1996.
[6] J.A. Brzozowski and C.-J. Seger, Asynchronous Circuits, Springer-Verlag, 1994.
[7] K.C. Tai, "Race analysis of traces of asynchronous message-passing programs," Proc. 17th Intern. Conf. on Distributed Computing Systems (ICDCS'97), Baltimore, MD, 1997, pp. 261–268.
[8] A. Bechini and K.C. Tai, "Timestamps for programs using messages and shared variables," Proc. 18th Intern. Conf. on Distributed Computing Systems, Amsterdam, May 1998, pp. 266–273.
[9] R.H. Netzer and B.P. Miller, "Optimal tracing and replay for debugging message-passing parallel programs," Proc. Conf. Supercomputing, 1992, pp. 502–511.
[10] K. Audenaert, "Maintaining concurrency information for on-the-fly data race detection," Proc. Parallel Computing 1997, Bonn, September, 1997, pp. 19–22.
[11] R. Alur, G. Holzmann, and D. Peled, "An analyzer for message sequence charts," Proc. 2nd Intern. Workshop TACAS'96, 1996, pp. 35–48.
[12] A. Petrenko, A. Ulrich, and V. Chapenko, "Using partial orders for detecting faults in concurrent systems," in: IWTCS'98, Tomsk, Russia, 1998, pp. 175–190.
[13] A. Ulrich and A. Petrenko, "Verification and testing of concurrent systems with action races," Proc. 13th Workshop "Test Methods and Reliability of Circuits and Systems", edited by Jurgen Alt, February 18–20, 2001, Miesbach, S. Bavaria, Germany.
[14] G. Luo, G. von Bochmann, A. Das, and C. Wu, "Failure-equivalent transformation of transition systems to avoid internal actions," Information Processing Letters, vol. 44, pp. 333–343, 1992.

Received following revisions 16 September 2002; originally submitted 2 January 2002